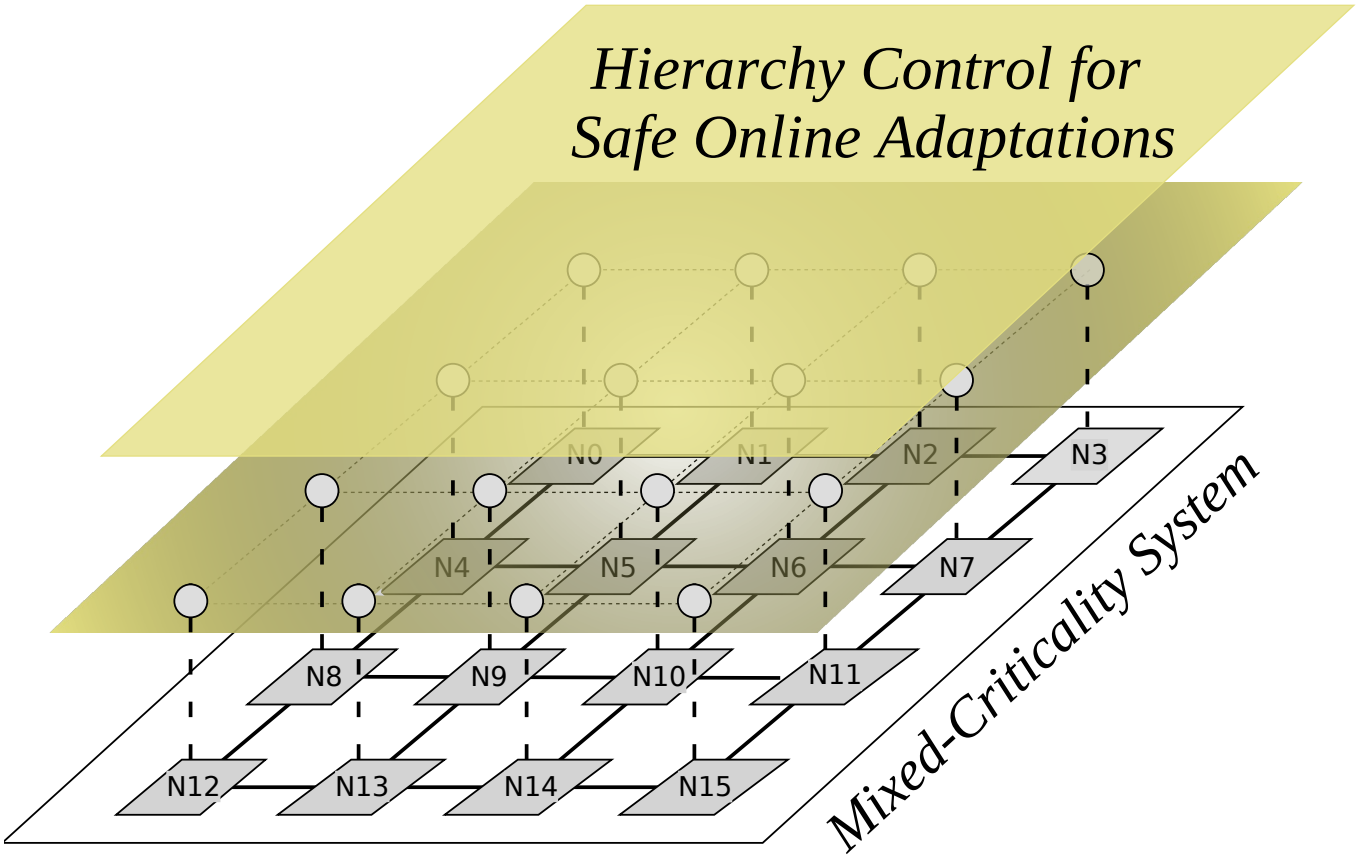
Dependable and Energy-Efficient Mixed-Critical Real-Time Systems-on-Chip



Thawra Mohamad Kadeed

Dependable and Energy-Efficient Mixed-Critical Real-Time Systems-on-Chip

Dissertation an der Technischen Universität Braunschweig,
Fakultät für Elektrotechnik, Informationstechnik, Physik

Dependable and Energy-Efficient Mixed-Critical Real-Time Systems-on-Chip

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades einer Doktorin

der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von Thawra Kadeed

aus Aleppo

1. Referent: Prof. Dr.-Ing. Rolf Ernst
2. Referent: Prof. Dr. sc. techn. Andreas Herkersdorf

Eingereicht am: 11.01.2021

Mündliche Prüfung am: 23.04.2021

Druckjahr: 2021

Abstract

Multi- and many-core embedded systems are increasingly becoming the target for many applications that require high performance under varying conditions. A resulting challenge is the control, and reliable operation of such complex multi-processing architectures under changes, e.g., changing environment, system dynamics, high temperature, and degradation. In mixed-criticality systems where many applications with varying criticalities are consolidated on the same execution platform, fundamental isolation requirements to guarantee non-interference of critical functions are crucially important, further intensifying the problem.

While Networks-on-Chip (NoCs) are the prevalent solution to provide scalable and efficient interconnects for the multiprocessing architectures, their associated energy consumption has immensely increased. Specifically, hard real-time NoCs must manifest limited energy consumption as thermal runaway in such a core shared resource jeopardizes the whole system guarantees. Thus, dynamic energy management of NoCs, as opposed to the related work static solutions, is highly necessary to save energy and decrease temperature, while preserving essential temporal requirements.

In this thesis, we introduce a centralized management to provide energy-aware NoCs for hard real-time systems. The design relies on an energy control network, developed on top of an existing switch arbitration network to allow isolation between energy optimization and data transmission. Consequently, our solution could be easily adopted by contemporary commercial NoCs as no modifications of routers are required. The energy control layer includes local units called *Power-Aware NoC controllers (PANCs)* that dynamically optimize NoC energy depending on the global state and applications' temporal requirements.

Furthermore, to adapt to abnormal situations that might occur in the system due to degradation, we extend the concept of NoC energy control to include the entire system scope. That is, online resource management employing *hierarchical control layers* to treat system degradation (e.g., imminent core failure) is supported. The mechanism applies system reconfiguration that involves workload migration. For mixed-criticality systems, it allows flexible boundaries between safety-critical and non-critical subsystems to safely apply the reconfiguration, preserving fun-

damental safety requirements and temporal predictability.

Simulation and formal analysis-based experiments on various realistic usecases and benchmarks are conducted showing significant improvements in NoC energy-savings and in treatment of system degradation for mixed-criticality systems improving dependability over the status quo.

Zusammenfassung

Eingebettete Many- und Multi-core-Systeme werden zunehmend das Ziel für Anwendungen, die hohe Anfordungen unter unterschiedlichen Bedinungen haben. Für solche hochkomplexed Multi-Prozessor-Systeme ist es eine grosse Herausforderung zuverlässigen Betrieb sicherzustellen, insbesondere wenn sich die Umgebungseinflüsse verändern, z.B. wegen sich verändernder Umgebung, Systemdynamik, hohe Temperatur oder Abnutzung. In Systeme mit gemischter Kritikalität, in denen viele Anwendungen mit unterschiedlicher Kritikalität auf derselben Ausführungsplattform bedient werden müssen, sind grundlegende Isolationsanforderungen zur Gewährleistung der Nichteinmischung kritischer Funktionen von entscheidender Bedeutung, was das Problem weiter kompliziert.

Während On-Chip Netzwerke (NoCs) häufig als skalierbare Verbindung für die Multiprozessor-Architekturen eingesetzt werden, ist der damit verbundene Energieverbrauch immens gestiegen. Insbesondere müssen harte Echtzeit-NoCs einen begrenzten Energieverbrauch aufweisen, da thermisches Durchgehen bei einer solchen gemeinsam genutzten Ressource die Garantien des gesamten Systems gefährden. Daher sind dynamische Plattformverwaltungen, im Gegensatz zu den statischen, zwingend notwendig, um ein System an die oben genannten Veränderungen anzupassen und gleichzeitig Timing zu gewährleisten.

In dieser Arbeit entwickeln wir energieeffiziente NoCs für harte Echtzeitsysteme. Das Design basiert auf einem Energiekontrollnetzwerk, das auf einem bestehenden Switch-Arbitration-Netzwerk entwickelt wurde, um eine Isolierung zwischen Energieoptimierung und Datenübertragung zu ermöglichen. Daher könnte unsere Lösung leicht von modernen kommerziellen NoCs übernommen werden, da keine Modifikationen von Routern erforderlich sind. Die Energiesteuerungsschicht umfasst lokale Einheiten, die als Power-Aware NoC-Controllers (PANCs) bezeichnet werden und die die NoC-Energie in Abhängigkeit vom globalen Zustand und den zeitlichen Anforderungen der Anwendungen optimieren. Zu diesem Zweck werden formale Verifizierungen vorgestellt, mit denen das Zeitverhalten sichergestellt werden kann.

Darüber hinaus wird das Konzept der NoC-Energiekontrolle zur Anpassung an Anomalien, die aufgrund von Abnutzung auftreten können, auf den gesamten

Systemumfang ausgedehnt. Online-Ressourcenverwaltungen, die hierarchische Kontrollschichten zur Behandlung Abnutzung (z.B. drohender Kernausfälle) einsetzen, werden bereitgestellt. Der Mechanismus wendet eine Systemrekonfiguration an, die eine Migration der Workloads beinhaltet. Bei Systemen mit gemischter Kritikalität erlaubt es flexible Grenzen zwischen sicherheitskritischen und unkritischen Subsystemen, um die Rekonfiguration sicher anzuwenden, wobei grundlegende Sicherheitsanforderungen erhalten bleiben und Timing Vorhersehbarkeit.

Experimente werden auf der Basis von Simulationen und formalen Analysen zu verschiedenen realistischen Anwendungsfallen und Benchmarks durchgeführt, die signifikanten Verbesserungen bei On-Chip Netzwerke-Energieeinsparungen und bei der Behandlung von Abnutzung für Systeme mit gemischter Kritikalität zur Verbesserung die Systemstabilität gegenüber dem bisherigen Status quo zeigen.

You should not look up what:
give up means,
or give in means.
Explore, analyse, and then, only then, give it all!

Thawra Kadeed

Contents

Chapter 1: Introduction

1.1. Problem Statement

Embedded systems are information processing systems embedded into enclosing products [111]. Seen or unseen, they are highly pervasive and everywhere, from kitchen appliances, vehicles, and aircraft to medical and military devices. Due to the ever-increasing demand for high performance together with low energy consumption and size, multi- and many-core embedded systems have emerged and rapidly grown. Multiprocessor System-on-Chip (MPSoC) enables complex functions and Electronic Control Units (ECUs) to be integrated in one chip, which previously were distributed (low performance) and communicated via external buses. This consolidation of functions with varying criticality levels results in mixed-criticality many-core systems [15, 54]. Mixed-criticality systems generally combine safety-critical and non-critical applications on the same execution platform. They are ubiquitous in cyber-physical systems with growing importance due to the aforementioned integration of complex function networks on fewer high-performance computing platforms, such as in automotive platforms for automated driving.

Figure 1.1 presents the vehicle computing transitions from many distributed ECUs, which have a significant synchronization and communication overhead, over a domain vehicle architecture to zonal vehicle architecture. In a domain or zonal vehicle architectures, multi-core ECUs are employed to provide the required functionalities. In addition, the domain vehicle architecture attempts to provide one ECU for each domain supporting domain isolation, however, the zonal vehicle architecture is a software-defined approach that processes the workload of different domains on the same ECU. The zonal approach obviously improves the communication between the processing elements and hence the performance. In turn, it reduces the isolation requirements as functions of different domains with diverse safety requirements are using the same resources, e.g. processors and network, which definitely requires thorough management to maintain isolation.

In particular, the autonomous driving systems must be highly reliable and guarantee sufficient functionality under platform errors, high energy consumption, ag-

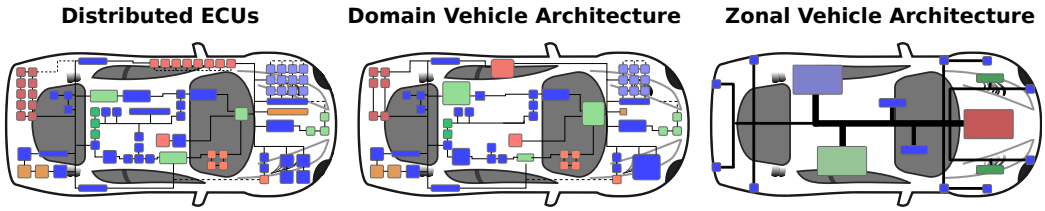


Fig. 1.1.: Automotive vehicle today vs. future [152]

ing, and degradation to meet safety requirements [81, 49, 78]. Figure 1.2 presents the responsibility transition from the human to the machine. There will be no driver supervision of the system's decisions and actions, consistently with [1]. Consequently, there will be no driver overtaking in case of errors, and the system must provide a technical fallback that requires to prove the proper behavior under all cases.

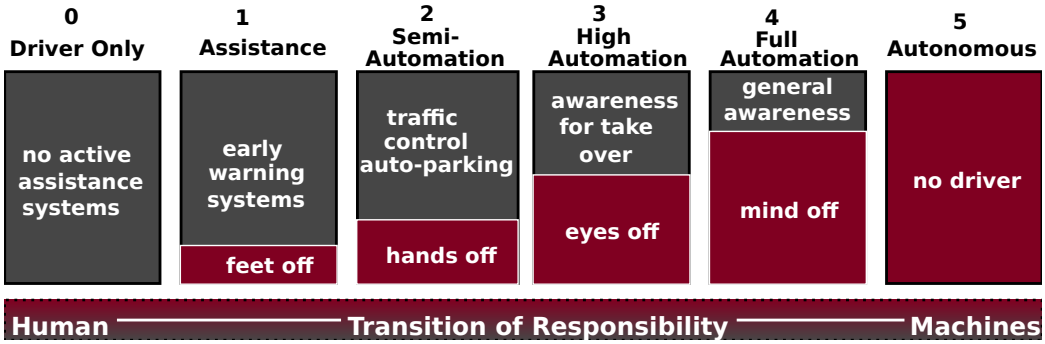


Fig. 1.2.: 5-layer shift paradigm towards autonomous driving [1]

Networks-on-Chip (NoCs) have replaced the bus structures in today's MPSoCs [25, 46, 151]. They provide a flexible interconnect for the complex multiprocessing architectures. They have been utilized to provide high performance [99, 101], quality of service [154, 152], and fault-tolerance [35, 136]. Figure 1.3 presents the NoC-based many-core system that is tile-based [153]. That is, the hardware architecture is organised in tiles. These tiles are connected by a network-on-chip. Network interfaces connect the tiles to NoC routers. Tiles with computing units are considered processing resources; the NoC, memory controllers, and I/O interfaces are shared resources.

NoC is a core part and a new aspect in MPSoCs with mixed-critical applications. It is a shared resource that hosts transmissions with varying criticality levels. Thus, NoC resource management is highly required. Several management metrics have been addressed, e.g., NoC performance [157], NoC fault tolerance [136], however, energy-aware NoC design for hard real-time mixed-critical systems is still an open problem. The energy consumption of networks-on-chip becomes tremendous in high density MPSoCs and thus increases the temperature which challenges the whole system guarantees. A NoC by itself consumes a substantial portion of total

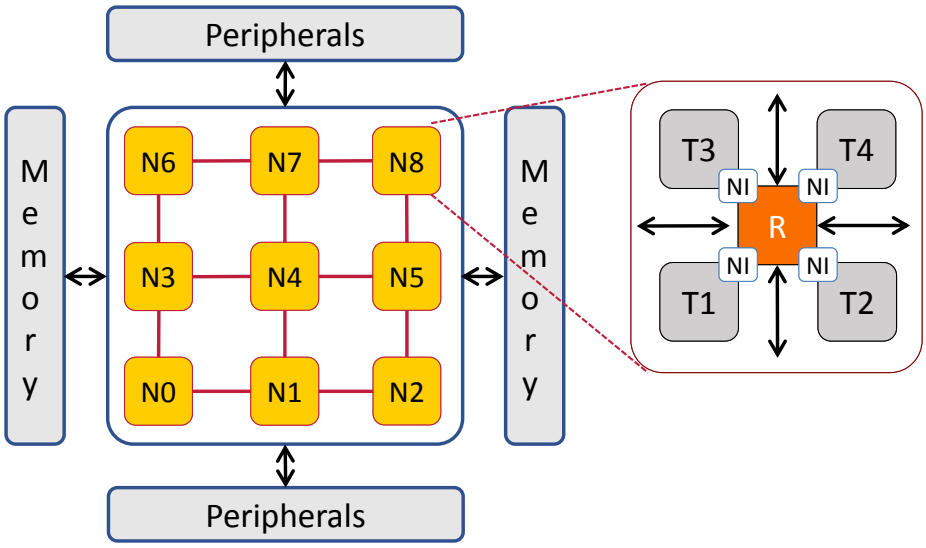


Fig. 1.3.: Architectural template of NoC-based many-core system-on-chip

chip power. For example, the router in Intel 80-tile Teraflop consumes up to 28% of tile power [76, 158]. Furthermore, the 16-tile MIT RAW NoC consumes 36% of total chip power, with each router dissipating 40% of individual tile power [150, 160, 47]. SCORPIO, a 36-Core research chip design, the Network Interface Controller (NIC) and the router consume 19% of tile power [45]. Other research work also reports that NoC consumes up to 35% of an overall chip power [38]. Especially in safety-critical hard real-time systems, dynamic energy management schemes of NoC must be developed to save energy and decrease the temperature in dynamic environments. In addition, NoC energy-savings must comply with temporal constraints where all firm deadlines of timing-critical functions must be met.

Furthermore, the problem is not going away, it is getting increasingly substantial when system abnormal operation is considered. That is, such a complex multiprocessor system is subject to change and throughout its lifetime, besides the threat of the usual transient, intermittent, permanent random hardware faults [61, 81, 72] and physical limitations such as dark silicon [56, 69], systems will face multi-dimensional variability, from varying workload, Quality-of-Service (QoS) goals, and non-functional requirements to varying environmental and operating conditions. Thus, besides the need for NoC energy management, system-level runtime management is also crucially important. In this work, we address the response to system degradation and allow hazard prevention. The treatment of degradation must first adapt the system to the current change before it continues operation. One of the serious sources of degradation is aging. For example, when

a processor core ages, it indicates a failure in the future. Therefore, the system must react to the cursors of degradation too early. Also, system-level runtime management must ensure guaranteed service under strict safety and availability requirements [81, 78, 49]. Especially in mixed-criticality systems, which will be employed in embedded MPSoC, the problem is more complex. This is due to the integration of safety-critical and non-critical functions on the same execution platform, where safety standards require non-interference of safety-critical functions (*freedom from interference* as in ISO26262 [81] or *sufficient independence* as in IEC61508 [78]). Consequently, any change to the system must preserve the required isolation between functions with different criticality levels.

1.2. Safety Standards

System safety is an essential requirement of the automotive, railway, avionic industries. To ensure system safety there are several research and industrial efforts towards standardization of electronic/electric products. There are many international organizations, which publish design guidelines and regulations for different domains. Some of these are the International Electrotechnical Commission (IEC) [78], and the International Standards Organization (ISO) [81]. The IEC published the generic standards for all systems that include electronic/electric elements. These standards could be optimized to feature specific domains, e.g., railway, machinery, power drive, as sketched in Figure 1.4. Based on IEC61508, Do254 and Do178B respectively published the design and validation standards for avionic hardware [49] and software [48], and ISO26262 has specified the standards for automotive domain [81].

ISO26262 introduces the V-Model design process for electronic/electric systems accounting for safety as depicted in Figure 1.5. It is based on a V-Model as a reference process model for the diverse phases of product development. It is a top-bottom approach that flows from specification of safety requirements, product development, implementation, test and verification, to production and operation. The shaded "V"s represent the interconnection between the phases. For example, the system development process is based on the specification of the technical safety requirements, the system architecture, the system design and implementation on the left side and the integration, verification, safety validation and the functional safety assessment on the right side [81].

Besides, safety lifecycle defines the required safety activities and their order from concept to decommissioning. The safety lifecycle elaborates and constrains the V-Model design process. Figure 1.6 illustrates the safety lifecycle introduced

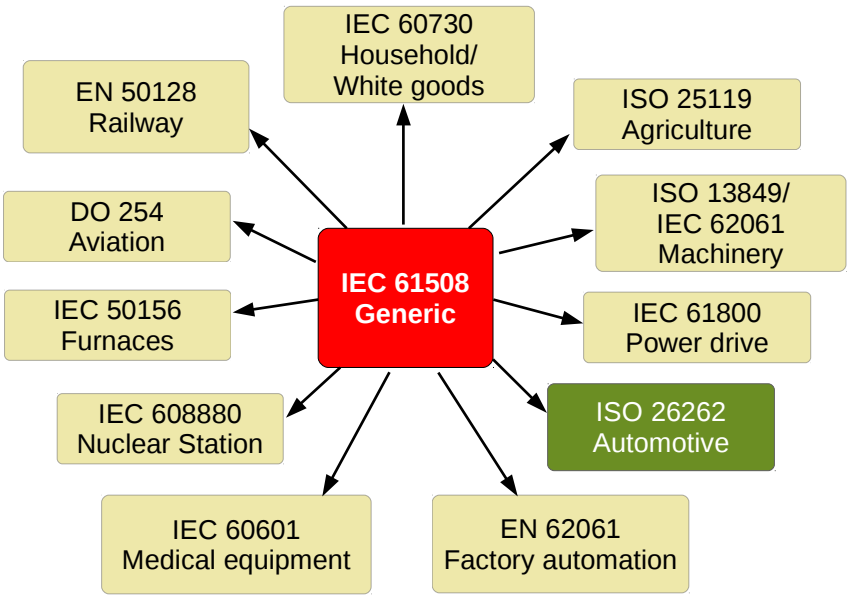


Fig. 1.4.: Domain specific safety standards extracted from IEC61508

by ISO26262 [81]. We are concerned in this work with three blocks that are highlighted in red, product development, controllability and operation planning. The concept phase branches to controllability unit that must manage and control the potential hazard reported by this phase before it becomes a serious risk and leads to failure. The controllability unit goes back to safety validation at which all safety requirements, derived at the concept phase, must be assured after the product is developed and before its release for production. In addition, the product development block branches to operation planning, which adds responsibility for planning of system operation accounting for safety. That is, it includes planning of the safety activities through the safety lifecycle which is considered a key management task [81].

The ISO26262 introduces safety requirements that must be fulfilled during the design process, e.g. *functional safety*, which is defined as *"the absence of unreasonable risk due to hazards caused by malfunctioning behaviour of electronic/-electric systems"* [81]. This depends on the criticality of system components. The ISO26262 defines the automotive safety integrity levels (ASILs) A through D plus quality management (QM), where QM is not safety-critical, A is the least critical, and D is the highest criticality level. All safety-related components must be developed to the highest ASIL. However, mixed-criticality is particularly difficult because it must ensure that functional and non-functional requirements of higher criticality functions are met while co-existing with less well-behaved functions of lower criticalities.

This is stated by ISO26262 "*If the embedded software has to implement software components of different ASILs, or safety-related and non-safety-related software*

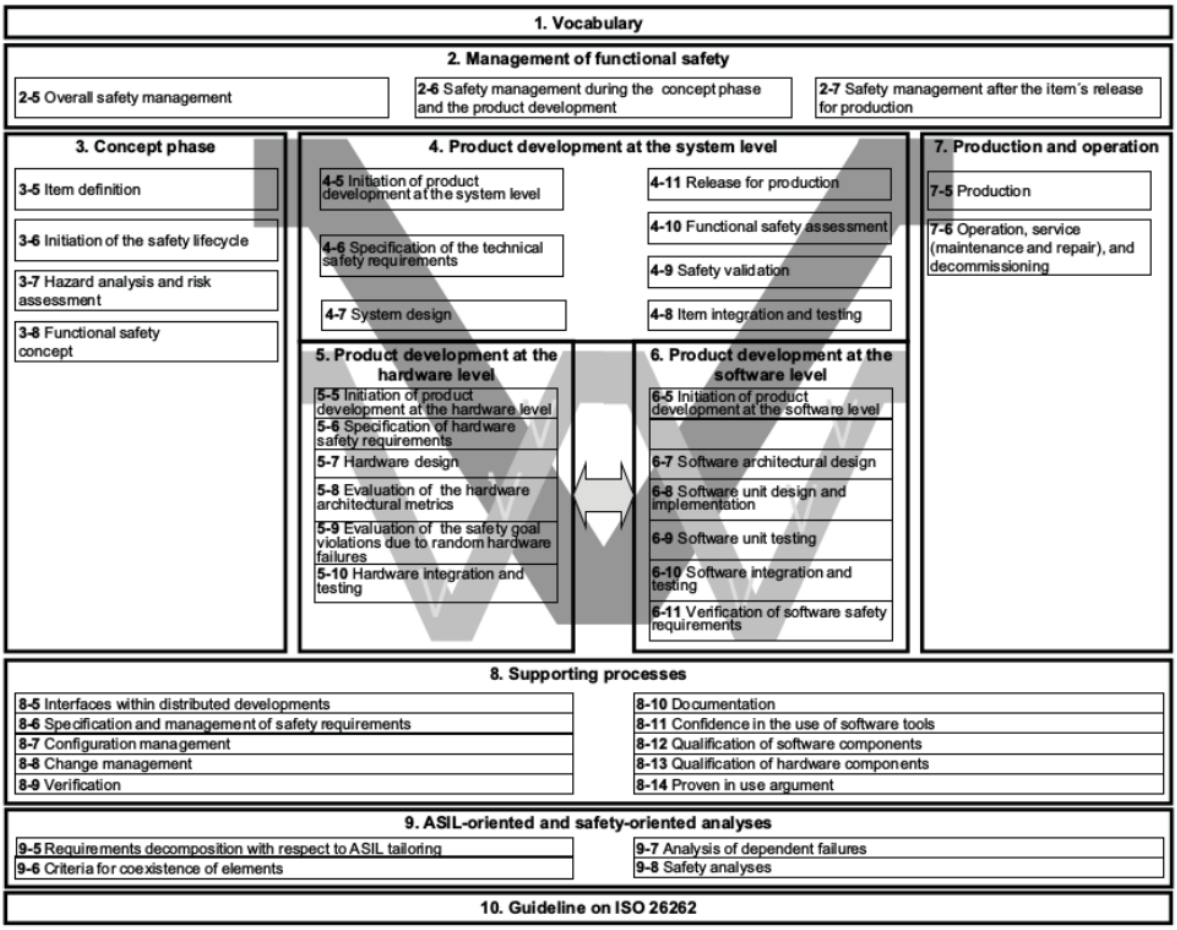


Fig. 1.5.: V-Model design process according to ISO26262 [81]

components, then all of the embedded software shall be treated in accordance with the highest ASIL, unless the software components meet the criteria for coexistence in accordance with ISO 26262-9:2018, Clause 6.", where Clause 6 proposes: in the case of the coexistence of sub-elements that have different ASILs assigned or the coexistence of sub-elements that have no ASIL assigned with safety-related ones, it can be beneficial to avoid raising the ASIL for some of them to the ASIL of the element. When determining the ASIL of sub-elements of an element, the rationale for freedom from interference is supported by analyses of dependent failures focused on cascading failures. The *freedom from interference* is also defined as *"the absence of cascading failures between two or more elements that could lead to the violation of a safety requirement"* [81]. Same principle in IEC61508 (*sufficient independence* [78]).

Consequently, as described in Section 1.1, to apply runtime management schemes to mixed-criticality systems, it is highly required to preserve *functional safety* and *freedom from interference* requirements that are still not covered by the related

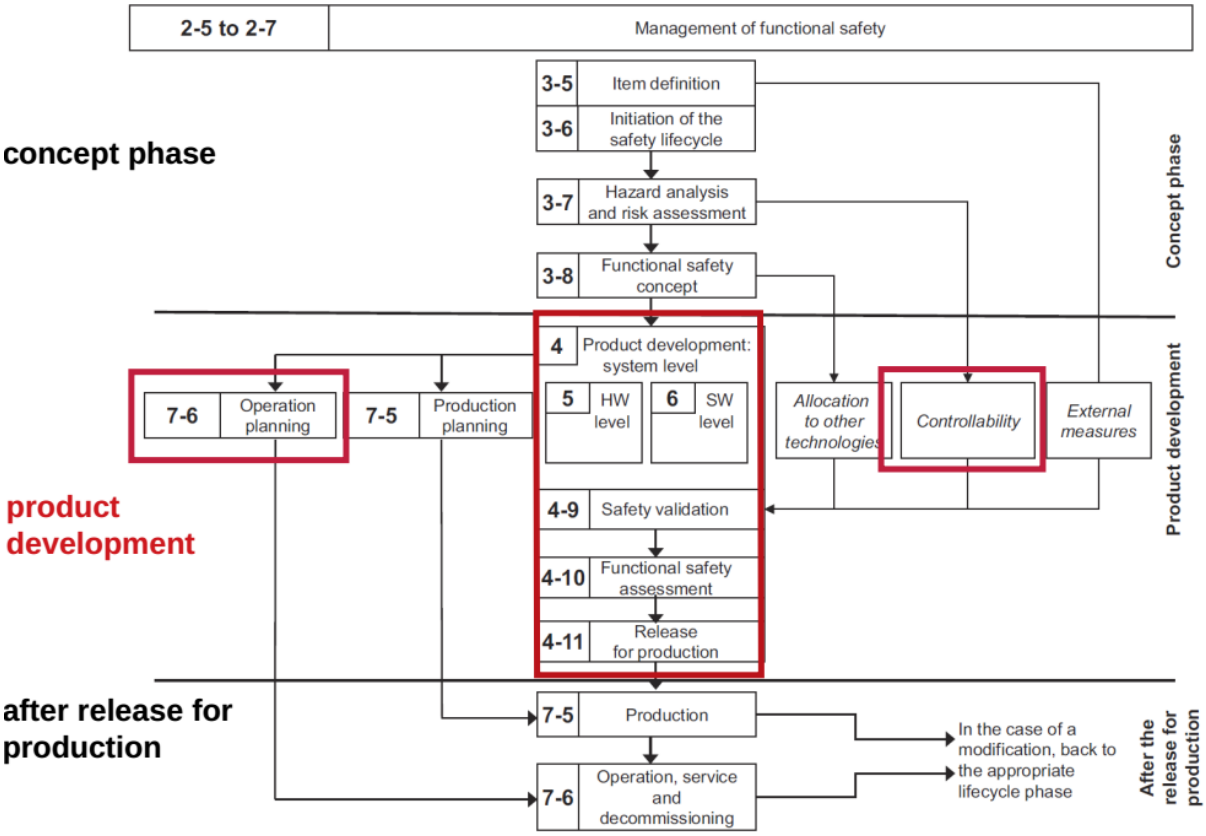


Fig. 1.6.: Safety lifecycle as introduced in ISO26262 [81] [53]

work. Besides, in most cases safety-critical functions are also subject to timing constraints. Due to their direct impact on functional safety, such systems are also strictly regulated by safety standards [78, 81, 49]. Thus, temporal guarantees must be also supported while runtime managements at NoC-level (energy optimization) and system-level (degradation treatment) are applied.

In the following, we present an overview of real-time systems with various temporal requirements and highlight hard real-time systems with firm deadlines.

1.3. Real-Time Applications Properties

Mixed-criticality embedded systems host heterogeneous applications with different requirements and behavior. This includes applications with different safety-criticality in addition to different temporal requirements. For example, automotive domain integrates object detection in Advanced Driver Assistance Systems (ADAS) and entertainment applications. Consequently, criticality can be divided into two distinct aspects, safety criticality and timing criticality. Safety-criticality corresponds to the integrity and proper functionality of the system (e.g.,

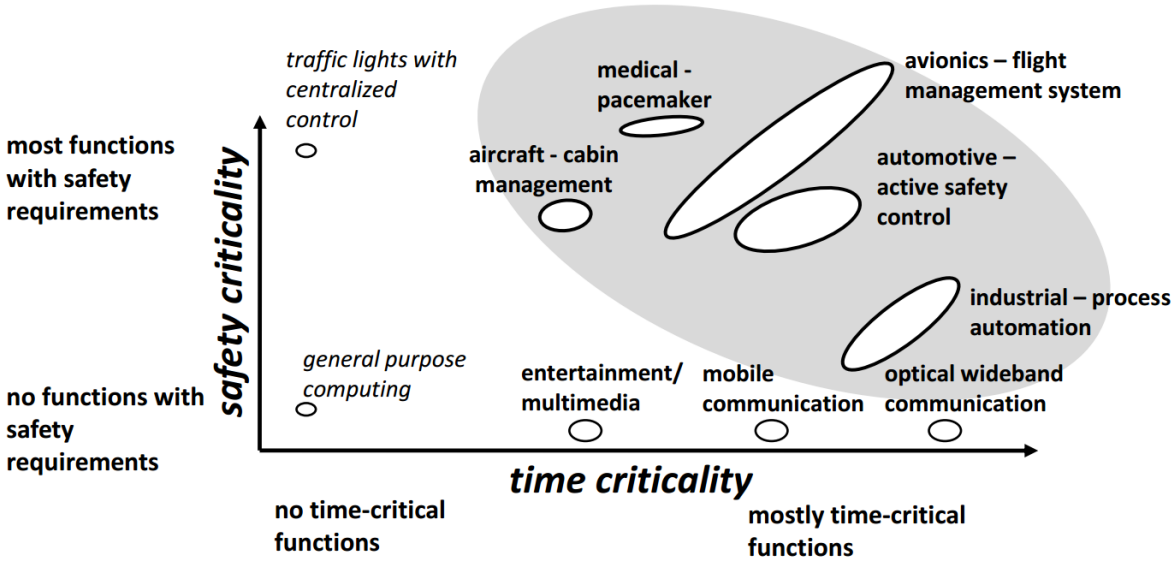


Fig. 1.7.: Two dimensions of mixed-criticality: Time and Safety [11]

traffic light). In turn, timing-critical applications must ensure the correctness of the system function by not only the proper behavior but also by assuring temporal guarantees. In other words, these applications must react within a precise time interval (e.g., breaking systems). The timing constraint by which a specific result must be produced is called deadline. Figure 1.7 depicts a general overview of varying application domains with different criticalities. As shown, not all real-time systems are safety-critical and vice versa. There are some systems with both safety and temporal requirements (e.g., engine control, pacemaker). These domains are of high interest as they target future platforms as in autonomous driving and thus must combine real-time and dependability techniques on the same execution platform.

The temporal properties of system applications can be classified into *best-effort*, *soft real-time*, and *hard real-time*. *Best-Effort (BE)* applications do not have strict temporal constraints, e.g, a deadline miss do not endanger system safety (e.g., entertainment applications). Consequently, temporal properties of such applications are less tested and their NoC accesses are not previously defined. These applications are designed targeting high performance and the system still need to provide sufficient resources to process them (e.g., be work conserving). Thus, achieving high performance is a common design goal as long as guaranteed service for safety-critical hard real-time applications is preserved.

In *Soft real-time* applications, the produced results have some utility after the deadline or are not influencing system safety [149]. For example, control algorithms based on a feedback loop or video analysis for night vision in a car. In such systems, the algorithms may tolerate limited cases when instead of new sampling

data, old values are employed [140, 156, 169].

Hard real-time applications have strict deadlines, i.e., the utility of the produced result is zero when the deadline is missed. In safety-critical hard real-time applications, a delay beyond a firm deadline results in severe consequences and endangers the whole system safety. Consequently, such applications must rigorously meet their deadlines, where the worst-case latency must stay below the firm deadline. In safety-critical systems, the correctness of the behavior, even in the worst-case, must be verified according to the standards [81, 78, 49]. Due to these requirements the characteristics of safety-critical hard real-time applications and their network accesses are well specified and tested and thus known at design time.

Hard real-time networks-on-chip host critical applications whose temporal constraints must be satisfied. The authors in [152, 97] present the requirements of such networks to achieve this goal and especially when the NoC host applications with different criticality levels (best-effort and safety-critical real-time senders), so-called mixed-critical NoC, which is employed in this thesis. Consequently, as stated above in Section 1.1 any change to these NoCs to optimize energy or prevent system failures must satisfy temporal requirements and maintain functional safety.

1.4. Research Objectives and Contributions

Many-core embedded system platforms running mixed-criticality applications have grown considerably in complexity, and they continue to grow. While networks-on-chip are the prevalent solution to provide a scalable interconnect for such complex multiprocessing architectures, their associated energy consumption has immensely increased. Therefore, efficient energy management for NoCs is especially important to save energy and decrease the temperature in safety-critical hard real-time systems, where capabilities of power supplies are additionally limited. NoCs for future mixed-criticality applications in dynamic environments require predictable and online energy management to adapt to dynamic system behavior. Besides, NoC reconfiguration to enforce new energy setups must adhere to timing constraints in hard real-time systems. The related work solutions typically induce a high timing overhead [112, 58], thus challenging timing safety, or has limited energy-savings capabilities [167, 168, 39].

We solve this problem by introducing a centralized NoC management to dynamically manage energy dissipation in hard real-time NoCs, overcoming static energy management in the status quo [167, 168]. The centralized function/decision is something new to safely control the NoC. It is composed of a global manager

and distributed local (on-tile) supervisors that work in synergy to provide efficient and predictable energy-savings for NoC routers. The approach exploits part of application properties like deadline slacks of timing- critical functions (time budgets are still available); and Acquisition-Execution-Restitution (AER) task model [50] to save energy under timing constraints. The AER defines a predictable access pattern of a safety-critical task by decomposing it into three consecutive phases: acquisition (read), execution, and restitution (write) where the approach saves energy during the task execution phase.

Furthermore, the problem including NoC energy control under system normal operation is more complex under system-level abnormal situations due to degradation. Especially in mixed-criticality systems, this is of high importance as any misbehaviors of untrusted/uncertified best-effort part might dramatically impact the proper functionality of safety-critical functions. As mentioned earlier, this is forbidden by the standards [81, 78]. Thus, complex systems-on-chip require dependable operation even under degradation. To this end, system reconfiguration must be enforced to adapt the system to the recent change and treat degradation, e.g., an imminent core failure. Also, the reconfiguration schemes must comply with safety requirements in mixed-criticality systems. System reconfiguration approaches are employed by the related work through applying migration of the critical functions from a failing core to a healthy core [146, 42, 127]. However, mapping reconfiguration of safety-critical hard real-time functions in mixed-critical systems is still an open problem as fundamental requirements including temporal guarantees, freedom from interference between critical and non-critical subsystems, protected accesses to shared resources, and robust reconfiguration are not satisfied.

To tackle this, we extend the concept of the centralised energy management of NoC to system-level degradation management by including treatment of imminent system failures. We achieve these goals via supporting a centralised reconfiguration manager in regular operation (NoC energy optimization), and hierarchical reconfiguration managers in abnormal behaviors (foreseen core failures) enabling planning of system operation accounting for safety. Such an approach allows predictability where system states are analysable and controllable which enables temporal and functional verifications required by the standards [81, 78, 49]. By doing so, we improve the system properties like NoC energy consumption and system dependability without challenging safety.

To fulfill the aforementioned objectives and tackle the respective challenges, **the contributions** of this thesis are fourfold:

- ***First***: We propose efficient and centralised management that allows global

and online NoC energy management under normal system operation. The approach introduces a control layer to save energy on the NoC data layer through multiple Power-Aware Network Controllers (PANCs) and local on-tile supervisors (clients). The approach exploits some system properties like task deadline slacks and AER task model to save more energy. Besides, we explore through PANCs the potential efficiency of integrating multiple energy-savings schemes, e.g., *Power-Gating (PG)*, *Clock-Gating (CG)*, *Dynamic Voltage and Frequency Scaling (DVFS)*, in the face of the diversity of energy dissipation sources (leakage, clock-tree, and dynamic).

- ***Second***: To safely apply the PANCs in hard real-time systems while assuring temporal guarantees, i.e., all deadlines of critical functions are met, formal worst-case analysis frameworks of the additional latency overhead induced by the control layer, including the timing induced by applying *PG*, *CG*, and *DVFS*, are provided. These frameworks allow to evaluate the feasibility of the control layer and verify temporal guarantees early at design time extracting the task slacks that are used later by PANCs at runtime. In addition, simulation and formal analysis-based experiments are conducted to respectively evaluate energy-savings in the average and the worst-case scenarios. The energy-aware NoC design is presented from concept to place and routed layout employing ASIC design flow and extracting efficient and accurate energy figures.

- ***Third***: We extend the concept of the NoC control layer optimizing energy to system-level hierarchical control layers to treat degradation. We present and evaluate an online reconfiguration approach that adapts the system to the current change and continues operation only after the risk is resolved, improving dependability. The system is supported by adaptive and proactive hazard prevention protocols to provide proactive actions that are particularly suited to treat foreseen system failures. These actions are guided by local controllers and last supervised by a global controller (system controller). They correspond to mapping reconfiguration of endangered safety-critical workload.

- ***Fourth***: To satisfy substantial requirements in mixed-criticality systems during reconfiguration including temporal guarantees, isolation between critical and non-critical subsystems, protected accesses to shared resources, and deadlock-free reconfiguration, a formal analysis framework and planning of safety activities are supported. Consequently, this allows flexible boundaries between best-effort and safety-critical subsystems, supporting continuous real-time system operation as required in many application domains. Besides, by conducting

simulation and analysis-based experiments, the enabled predictable online reconfigurations induce tolerable latency overhead.

To the best of our knowledge, this is the first work that investigates efficient energy-savings through integrating multiple energy-savings schemes, while simultaneously supporting temporal guarantees for hard real-time NoCs deployments in safety-critical domains (e.g., automotive). In addition, the work extends its scope to treat system degradation and resolve the risk of imminent failures, while preserving strict temporal and safety requirements in mixed-criticality systems.

1.5. Structure Overview

This thesis is organised as follows. Chapter 2 presents the main concepts of the energy control layer to manage and optimize NoC energy. Besides, an overview of the hierarchical control layers extending the scope of NoC control layer to system-level degradation management is introduced. Chapter 3 presents a power analysis framework and demonstrates the power figures of NoC routers in hard real-time systems. In addition, the individual/joint application of diverse energy management schemes (*Power-Gating, Clock-Gating, and Dynamic Voltage and Frequency Scaling*) is proposed to allow efficient energy-savings of NoCs. Chapter 4 provides formal analysis frameworks to demonstrate the applicability of the introduced control layer for hard real-time NoCs, where all tasks are characterized by their firm deadlines. Besides, the evaluation of the energy control approach through formal analysis and simulation-based experiments employing synthetic workload, benchmarks, and realistic usecases from automotive and avionic domains is investigated. Chapter 5 presents online adaptations to changes in mixed-criticality systems. A hierarchical control approach that provides online reconfigurations, which are particularly beneficial for treating foreseen system failures in many-core platforms is proposed. In addition, to allow safe online reconfigurations, fundamental safety and temporal targets are preserved. Experimental evaluations to investigate the respective overhead of the online reconfiguration are conducted employing automotive exemplar. Finally, Chapter 6 concludes with addressing envisioned future work.

Chapter 2: Safety-Compliant Runtime Management

Modern computers exploit parallelism throughout the system stack to increase performance and efficiency [71]. The results are computer systems with complex multiprocessing architectures whose control, optimization, and reliable operation have become significant challenges. As identified earlier, there are two crucial issues with which we concern, (i) NoC energy optimization decreasing temperature, (ii) system-level degradation treatment improving dependability. This chapter gives a closer look at the proposed solutions employing centralised management units. Figure 2.1 illustrates a global overview of a NoC-based many-core platform that is supported with our centralised controllers. The platform is organised in two domains, the execution domain, and the supported control domain. We distinguish between two zones, the safe zone (hosting safety-critical applications) and the unsafe zone (hosting best-effort applications). The control domain manages the platform targeting both NoC energy optimization (the PANC), and degradation control at the system-level (the system controller).

The PANC is introduced through a control layer to thoroughly optimize NoC energy. It dynamically manages energy accounting for the workload dynamic behavior. It employs the actual system global state to decide at runtime whether it is safe to save energy complying with temporal properties. The approach applies a protocol-based NoC access control through employing local supervisors and predictable PANCs. It is based on the concept of the NoC Resource Manager (RM), which has successfully been applied at runtime to reach high-performance in real-time systems [100, 101]. The detailed description of NoC energy control employing PANCs, formal verifications in terms of temporal guarantees, and experimental evaluations is presented in Chapters 3 and 4.

Furthermore, as the complexity of embedded system platforms used in mixed-criticality applications is rapidly growing, system-level dependable operation and long lifetime are required. To this end, the system controller extends the concept of the PANC to the system level. Online reconfigurations are supported via coordination between the system controller and distributed local managers to par-

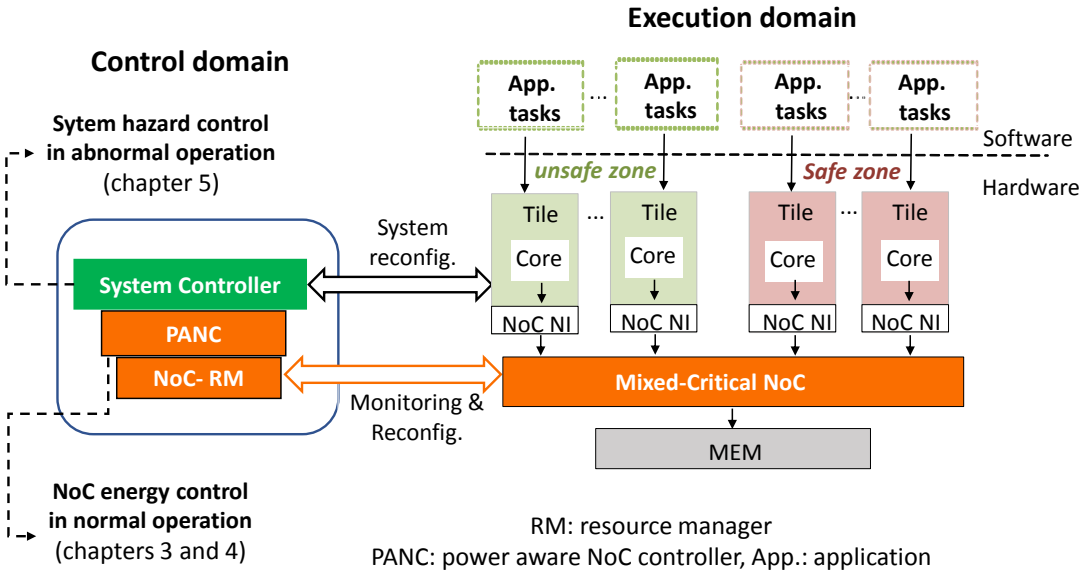


Fig. 2.1.: NoC-based many-core systems supporting NoC and system levels controllers

ticularly treat foreseen core failures, while preserving strict safety and temporal requirements. The detailed functionality of the hierarchical managers that are orchestrated by the system controller is introduced in Chapter 5.

This chapter first presents an overview of networks-on-chip architecture specifying hard real-time NoCs as introduced in Section 2.1. The main concepts of the energy control layer required to manage and optimize NoC energy are introduced in Section 2.2. The extension of NoC control layer to adapt to system severe changes is merely introduced in Section 2.3. Finally, Section 2.4 concludes this chapter.

2.1. Networks-on-Chip

Networks-on-chip offer scalability, modularity, high data throughput, enhanced performance, and design productivity in multi- and many-core systems-on-chip [43, 5, 18, 19]. NoC, as an important shared resource, handles all communication entering, leaving or occurring within the chip, e.g., I/O operations and memory transactions.

There exist several properties of networks-on-chip design. For example, the *NoC topology*, the *routing algorithm* for moving data though the NoC, the *switching technique*, and the *flow control*. A designer has diverse potential strategies to design a NoC router. This is the primary advantage of the NoC interconnect and explains why it has more chances to meet the communication requirements. Different from a bus structure, one can flexibly design a NoC according to system requirements. In the following, we summarise the major properties of networks-

on-chip [41, 43, 3].

2.1.1. Topology

NoC topology defines how the network routers are connected to each other, i.e., the structure of the routers connections. This impacts directly the system performance, energy consumption, and hardware overhead. Topologies are classified into direct or indirect, and regular or irregular. The direct topologies indicate that every router is associated to a processing element and this pair appears as a single element in the system. In this topology, each element is directly connected to a fixed number of neighbour elements. Examples of direct topologies are 2D mesh, torus, and the hypercube. In contrary to direct topology, not all routers in indirect topologies are connected to a processing element. Some routers are used only to forward the message. Some common examples for the indirect topologies are star, binary-tree and butterfly fat tree.

Furthermore, within regular topologies, routers follow similar connection patterns and are identical in the number of ports connecting to other nodes in the network. Examples are 2D mesh regular topology, ring, and 2D torus [41]. In irregular topologies, routers may present different connection patterns that are usually defined by the application, e.g., irregular 2D mesh topology. Figure 2.2 depicts some common NoC topologies [41, 43, 3, 152].

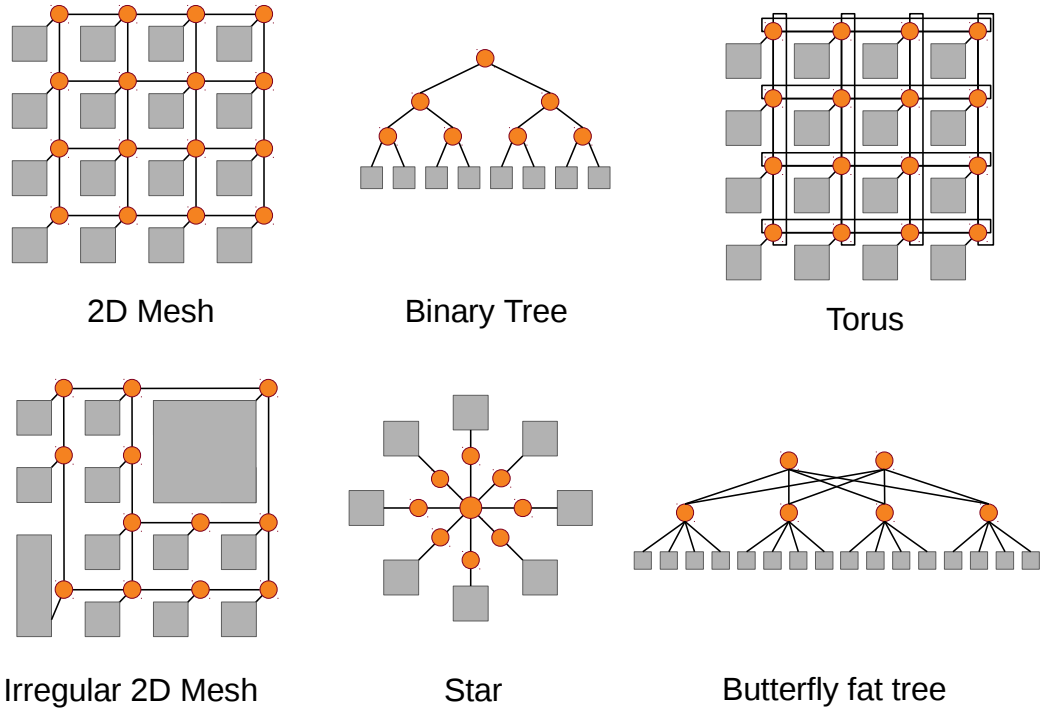


Fig. 2.2.: Common topologies for NoC structure

2.1.2. Routing Algorithm

The routing defines the strategy to select a path from the sender to the receiver within a network [43]. The routing algorithms of NoCs can be mainly classified into *Deterministic* and *Adaptive* algorithms. In *Deterministic* routing, the route between the source and the destination is fixed and is not affected by the network state. Examples are *XY* routing and *XYX* routing [36]. For *Adaptive* algorithms, the routing path is dynamically computed at runtime based on the current network state. In addition, routing can be implemented at the source tile, and thus called source routing, or distributed in the network. For source routing, the route information is stored at the source tile (sending node). It is usually stored in the packet header. Then, each router selects the correct output port based on this information. On the other hand, for distributed routing, only the address of the destination node is stored in the packet header. Each router then computes the corresponding output port based on this address.

2.1.3. Switching Technique

The switching defines how the traffic is transmitted through the NoC routers from the source node to the destination one. There are various switching techniques, *circuit switching*, and *packet switching*. In the *circuit switching*, the entire path from the source to the destination is allocated for the date transmission before it is sent. After the route is established, the payload can be pipelined along the route through the network. On the other hand, in the *packet switching* the message is divided into packets that can be sent independently through the NoC without the need to allocate the entire path at once.

The *packet switching* can distinguish three various techniques for buffering and forwarding, *Store and Forward*, *Wormhole*, and *Virtual Cut Through* switching. With *Store and Forward* technique, a complete packet is stored in the router buffer before the connection to the next router is established. That in turn requires sufficient buffer space to accommodate the packet size. In the *wormhole* switching, the packets are further divided into smaller units called Flow Control Units (FLITs). Forwarding a flit to a next router is established once the flit is arrived and thus the switching is done at flit level. The flits are pipelined and sent independently. In case of congestion at a certain link, the blocking is propagated back to all other links where all other flits are also blocked. The *Virtual Cut Through* switching is very similar to the wormhole technique, except that a congestion influences only the current link. That is, before forwarding the first flit to the next router, the precedent router waits for an acknowledgement that the entire packet can be

accep ted form the next router. Thus, other links can complete sending the packet flits without being influenced by the congestion.

2.1.4. Flow Control

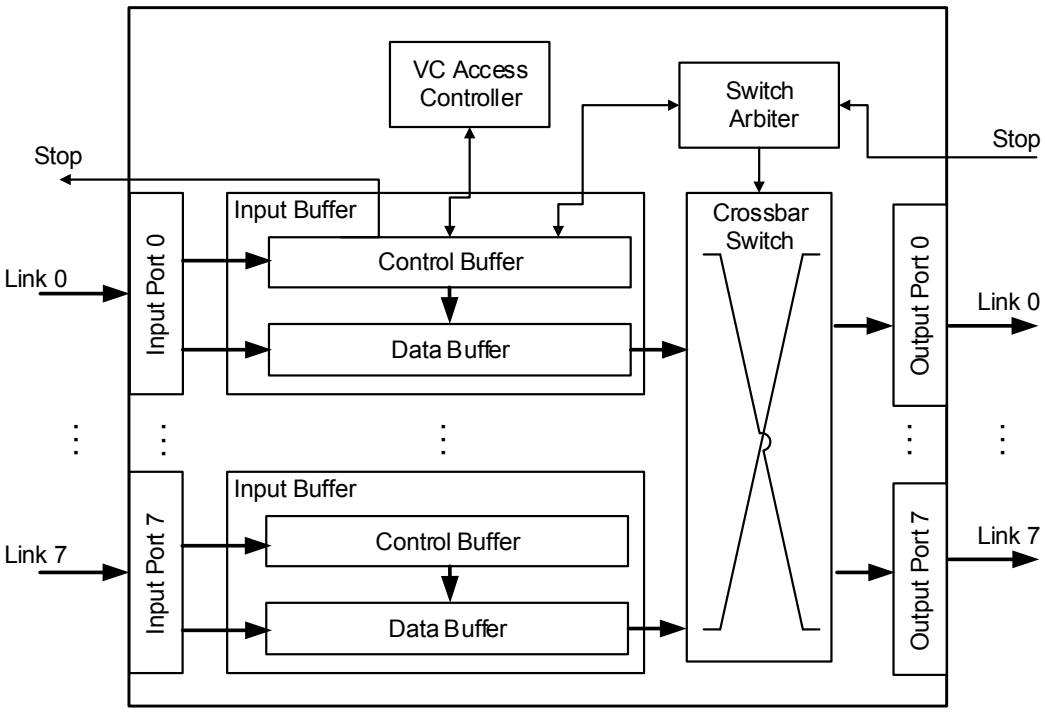
In order to ensure deadlock-free routing, prevent loss of packets or flits, and thus guarantee reliability of the transmission, flow control protocols are implemented through the network. It is needed to deal with congestions that lead to a buffer overflow. The flow control monitors the current permissible buffer space, such that data chunks are only transmitted when the needed buffer space is available or delayed until the buffer is free. Based on the used switching technique, flow control between NoC routers is performed at flit or packet level. Common flow control protocols are, *credit-based* flow control, *Ack/NAck* protocol, and *STOP-and-GO* flow control. In the *credit-based* flow control, the sender is supported with an amount of credits that are decreased when sending a flit. When the receiver processes a flit, it sends back a credit signal to the sender to indicate an increase of the available credits. On the other hand, in *Ack/NAck* protocol, the sender sends a flit, buffers it, and waits for an Acknowledgement (Ack) form the receiver. In case of an *Ack* signal is received, the next flit is sent, and when a NAck signal is received, the old flit is sent again. Eventually, with *STOP-and-GO* flow control, the receiver sends a Stop/Go signal to the sender. When a Go signal is received, the sender is allowed to transmit flits. However, a Stop signal indicates that the receiver currently is not able to handle incoming flits and the sender must stop transmitting.

2.1.5. Virtual Channels

Virtual Channels (VCs) are employed to split a physical channel in the network into multiple virtual channels. To implement VCs on a link the buffer on the link (i.e., in the ingress or egress port of the routers) has to be extended to include multiple (virtual) buffer queues, one for each VC. This way, buffers allocation is decoupled from allocating links when supporting multiple buffers for each link. Virtual channels mainly increase the hardware overhead needed for the router, but introduce some advantages. The main goal of a VC implementation is to improve system performance by avoiding deadlocks, optimizing link usage, and providing some traffic guarantees [41]. The concept of VCs allows more output paths per link, which reduces the potential interference of different streams for the same buffer.

2.1.6. Selected NoC Architecture

There exists a variety of NoCs available in academia as well as commercially. Examples of commercial MPSoCs that use NoCs are Kalray MPPA [25, 46], Tilera tile64 [151, 16], Arteris FlexNoC [6], and Netspeed's Orion [120] and Gemini [121].



(a) The block diagram of the architecture of the baseline NoC router in IDAMC

HF/SF	VC	Flit Type	Route	Tile Port	Payload
Size	3 bits	2	30	3	102

BF/TF	VC	Flit Type	Payload
Size	3 bits	2	135

(b) The flits formats including Single (SF), Head (HF), Body (BF), and Tail (TF) flits

Fig. 2.3.: Router architecture (a) and flits formats (b)

Our research vehicle is the IDA NoC, featured in the research platform Integrated Dependable Architecture for Many Cores (IDAMC) [153]. The IDA NoC is a NoC for hard real-time mixed-criticality systems. The NoC is wormhole-switched, where variable-sized packets are divided into fixed-sized FLITs, which are transmitted through a number of VCs; implements deterministic XY source routing, where the route and VC are defined at the source node; and supports VC flow

control with Stop-and-Wait mechanism per virtual channel. The NoC routers implement SLIP based arbitration [113], consisting of a two stage round robin. The block diagram of the router architecture is shown in Figure 2.3a. The routers are input-buffered and each input buffer is composed of 5 VCs, where each VC is a FIFO queue. The Virtual Channel Access Controller (VCAC) manages the access to VCs in downstream routers. The switch arbiter implements the SLIP arbitration and gives the arbitration grants to the crossbar switch that connects the input buffers to the respective output ports. After that, the flits are transmitted directly from the input buffer to the downstream router.

The router implements stop-and-wait flow control, raising a stop signal to inform the near-full state of the VC queues to the upstream router. Moreover, before the flit is stored in the input buffer, route management is performed to keep the routing data for the downstream router always in the first position. The route is encoded as a list of runs, where a run is a pair of output port (direction) and number of hops (distance). During a run, the hop counter is decremented. Once the packet finishes a run, the route is rotated. That is, the finished run is moved to the end of the list and the next run becomes the head. Our analysed router comprises 4-stage pipeline. The packets are composed of a Head Flit (HF), zero or more Body Flits (BFs), and one Tail Flit (TF). A Single Flit (SF) packet is also supported. The flits formats are depicted in Figure 2.3b. HFs and SFs (BFs and TFs) are identical except for the flit type, which distinguishes their semantics. For transmission, a flit is further divided into Physical Units (phits) to match the link bandwidth (i.e., 4 phits of 35 bits).

We focus on hard real-time mixed-criticality NoCs, where both timing-critical and best-effort applications co-exist in the same platform. In the following, we present the detailed concept of the centralised management using the control layer, architectural view and its integration with the existing NoCs, forming a new NoC design for energy management.

2.2. NoC-Level Control Layer Integration

Complex application networks are increasingly integrated in safety-critical real-time systems such that in automotive or avionic domains. Networks-on-chip designed to host advanced embedded applications require efficient real-time guarantees under tight power limitations. This section presents the main concepts of the control layer approach that allows energy-aware NoC design, while assuring timing predictability so that guarantees of hard real-time tasks are given. The control layer implements a protocol-based synchronization to know the global

state of the (part of) NoC, and applies this knowledge to save energy while keeping the system predictable. To this end, each sender synchronises first its NoC access with the PANC, and waits for the respective Ack. The PANCs units grant the senders the NoC access based on their real-time requirements and the routers states (energy modes). This implies an additional delay for each task access to the NoC. However, to ensure adherence to real-time requirements, the additional timing overhead must be bounded. In other words, an analysis should provide, at design time, the timing overhead of the control layer from which the safe application of the PANCs can be derived (as introduced in Section 4.1). Besides, the permissible slacks of critical functions are extracted (at design time) and exploited by the PANCs (at runtime) to apply the potential energy-savings schemes – providing application temporal guarantees.

The functional perspective of the PANC is based on the global state of the NoC, e.g., the concurrent active tasks from which the PANC derives the required energy settings of some routers/whole NoC. Each sender is provided with a local supervisor called client that supports the PANC with the current state of the sender (request NoC access/release NoC resource) via fulfilling a synchronisation protocol. To specify the proper design choice of the control layer for NoC deployment in safety-critical hard real-time systems, we present in the following the general requirements and objectives that must be satisfied.

2.2.1. Requirements

For the design of the core modules of the control layer, the local supervisors, the clients, and the control unit, the PANC, several aspects and requirements must be accounted for. First, the non-functional requirements including the real-time constraints in safety-critical hard real-time systems must be assured. Second, the functional requirements of the PANC and clients must be derived and satisfied. The real-time requirements are considered and hard deadlines are guaranteed as discussed later in Section 4.1. Moreover, the integrity requirement of not loosing or rerouting data through the network must also be satisfied. Integrity is defined as the absence of improper system alterations [10]. All errors and deviations from the proper behavior must be first detected and then contained before propagating through the system. The authors in [136] introduce guaranteed integrity and packet delivery of the transmitted data, i.e., the payload in real-time NoCs. Thus, in this work we focus on other aspects concerning timing and interference as clarified later.

On this basis, we can formulate the general design requirements of the PANC and clients modules. The functional requirements of the clients are summarised

as follows:

- Generating and issuing request messages.
- Processing of acknowledgement messages.
- Processing of configuration messages, e.g., stop and resume the current active tasks.
- Releasing NoC resource by detecting the end of a transmission, which also requires generating and issuing release messages.

Next to this, independently to the energy management scheme featured by PANC and its actual implementation, PANC must satisfy the following functional requirements:

- Receiving, qualifying and processing control messages by specifying the message's type, and the transmission's sender and receiver.
- Generating and issuing different various of control/configuration messages.
- Enforcing new energy settings on NoC according to the current system state and the employed energy management scheme.
- Assuring safe transitions between NoC energy-modes, preventing potential deadline misses due to sporadic overloads.

Thus, the actual implementation of the PANC and the clients must adhere to the presented requirements. Breaking the introduced functional and non-functional requirements down to the level of the network-on-chip, One essential requirement (*Req*) and three design objectives (*Obj*) of the control layer can be derived as follows:

- ***Req1:*** Temporal guarantees of timing-critical tasks. The control layer incurs additional latency overhead that might influence the temporal guarantees of safety-critical hard real-time tasks. Thus, the control layer latency overhead must be bounded and validated against the tasks hard deadlines. This requirement is satisfied through developing the respective timing analyses engines presented in Section 4.1.

- ***Obj1:*** Low latency overhead of control messages. As temporal overhead of the control layer has a direct influence on the applications temporal guarantees, reducing the latency overhead of the control messages, required by a synchronisation protocol, is indispensable.

- ***Obj2:*** Low interference of the control messages on data traffic. They must manifest low interference on existing traffic to minimize additional interference and blocking in the underlying network. *Obj1* and *Obj2* are satisfied through isolating the developed control layer from the existing NoC layer, as highlighted in Section 2.2.2

- ***Obj3:*** Low hardware and energy overheads. As the control layer is developed to feature energy-efficient NoC design, its respective area and energy overheads must be dramatically low. The experimental evaluations of the developed control layer, introduced in Section 4.2.3, indicate that the control layer dedicates only very small overhead of the NoC's die area and energy.

Besides the primary importance of low latency requirement that allows temporal guarantees, the hardware and power overheads are also significant. Networks-on-Chips are already complex systems and consume a considerable proportion of the chip area and power [24, 23]. For example, the network of Intel's 80-Core Teraflops processor consumes 17% of die area, where a router consumes up to 28% of tile power [76, 158]. In addition, the network buffers of the Intel's Teraflops consume 22% of the communication power [158]. Next to this, the iMesh network buffers of the TILE64 many-core system contribute by 60% to the NoC's area [162]. Likewise, the router of the TRIPS chip dedicates 75% of the router's die area for input buffers [64]. Overall, reducing additional overheads induced by the control layer design can significantly contribute to reduce the respective area and energy dissipation incurred by integrating our approach in NoC architecture.

The implementation of PANC and the clients may have different variants. The clients components can be implemented as hardware modules, extending the network interfaces to provide high performance or as software modules running on the cores to provide flexibility. In the same manner, PANCs can be fully realized in hardware, i.e., as an independent Intellectual Property (IP), or as a software running on the core. Isolating the implementation of the clients and PANC from applications running on processing elements is of high importance and especially in safety-critical real-time systems where certification is highly considered (fulfilling $Obj1$ and $Obj2$). In such systems, it is necessary to either certify the whole system (including all applications) to the highest relevant safety level, or providing sufficient independence as required by the standards [81, 78, 49]. We decouple the schemes developed for NoCs energy-savings from the traffic arbitration units and the senders, supporting the required sufficient independence. This way, we dedicate an additional interconnect to conduct control messages without influencing the regular date traffic (as detailed in the following section). Consequently,

adherence to standards can be achieved through clients and PANC that can be developed/designed to the highest relevant safety level.

2.2.2. Architectural Design

The essence of the control layer is composed of local and predictable PANC unit (or multiple ones in case of large NoCs), local supervisors (clients), and an interconnect. The general architecture of the control layer is depicted in Figure 2.4 [91, 90, 85]. PANC, as a global module, is responsible for energy-savings under real-time constraints. The clients components synchronise the physical NoC accesses of the senders running on the processing nodes (N) with PANC by employing additional direct links. The latter convey control messages (7 bits: 3 bits to convey the control message, 4 bits to determine the destination/source address in 4×4 NoC in case of request/release), and power/clock-state messages (1 bit) between PANC and the clients. The control messages are introduced later in Section 2.2.4. Note that we employ direct connections (links) in order to speed up the communication between PANC and the clients – mitigating the latency overhead of the control messages and satisfying *Obj1* (cf. Section 2.2.1). When, in large NoCs, in case of disjoint real-time applications, we split a NoC into multiple independent regions. Multiple PANCs can be implemented as each of them controls one NoC region. However, in case of applications where communications comprise several such regions, PANCs can communicate with each other using an additional interconnect (control network) in order to exchange each other regions' states, as demonstrated in Section 2.2.5.

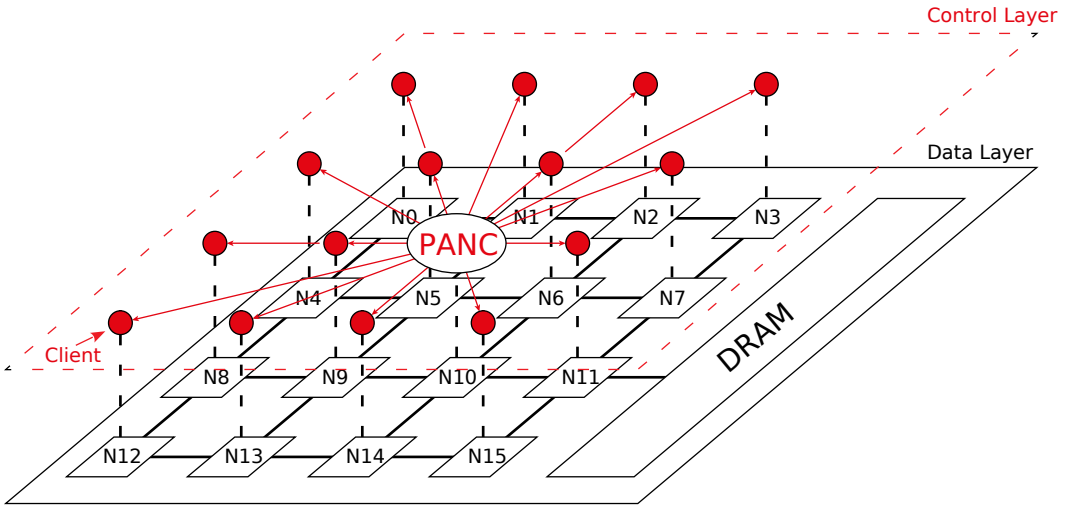


Fig. 2.4.: *PANC* within the control-layer (red color) for energy-savings of NoC data layer

The high-level view of the control layer integration in the NoC-based real-time systems is depicted in Figure 2.5. The architecture comprises three layers, the

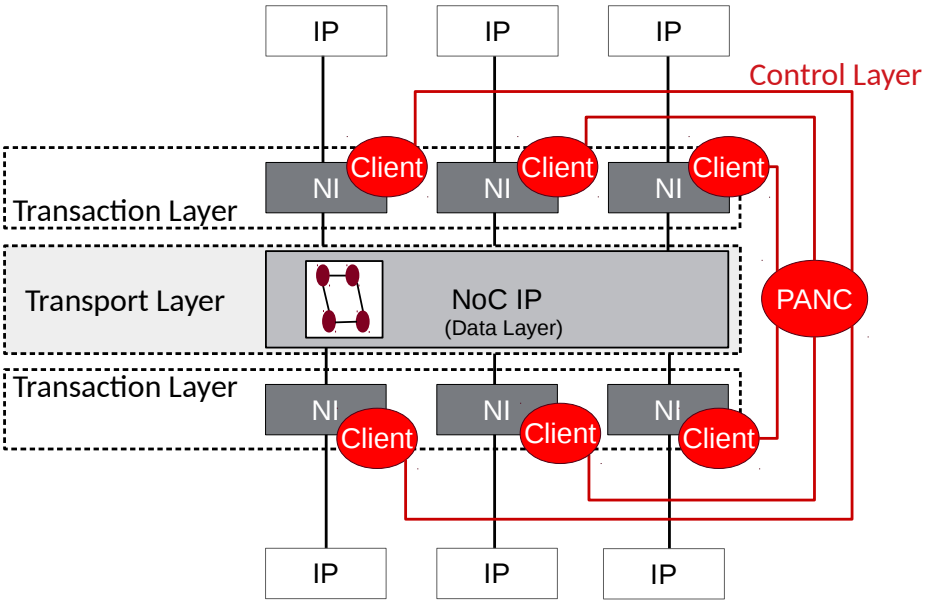


Fig. 2.5.: The integration of the proposed control layer (red color) in the NoC-based SoC, forming energy efficient NoC

interconnected IP cores, a transaction layer, and a transport layer. The IPs can be processing elements (PEs), memory controllers, or I/O interfaces. The transaction layer manages and controls the end-to-end transmissions and as well the synchronisation protocol for accessing NoC. Its primary components are the network interface (NI), connecting the IPs/cores to the NoC, the clients, constituting interfaces between the senders on the cores, and the PANC. The transport layer is a physical layer, responsible for moving the transmissions from one IP to another. The transport layer is extended by PANC as an independent module, responsible for NoC energy-savings, and the additional direct links that transfer the energy-oriented control messages between the clients and PANC. Thus, we clearly isolate the data messages conducted on the NoC data layer from the control messages conducted on the control layer, fulfilling *Obj1* and *Obj2* (Section 2.2.1).

In the following, we present a comprehensive framework of the control layer that orchestrates between runtime and design-time modules to perform dynamic energy management of hard real-time NoCs.

2.2.3. Comprehensive Framework

The control layer explores the potential energy-savings employing a comprehensive framework that incorporates dedicated analysis engines. The latter support the PANC with significant metrics required to negotiate the granularities of energy-savings, while assuring real-time requirements. Figure 2.6 illustrates the block diagram of the comprehensive NoC energy control framework, employ-

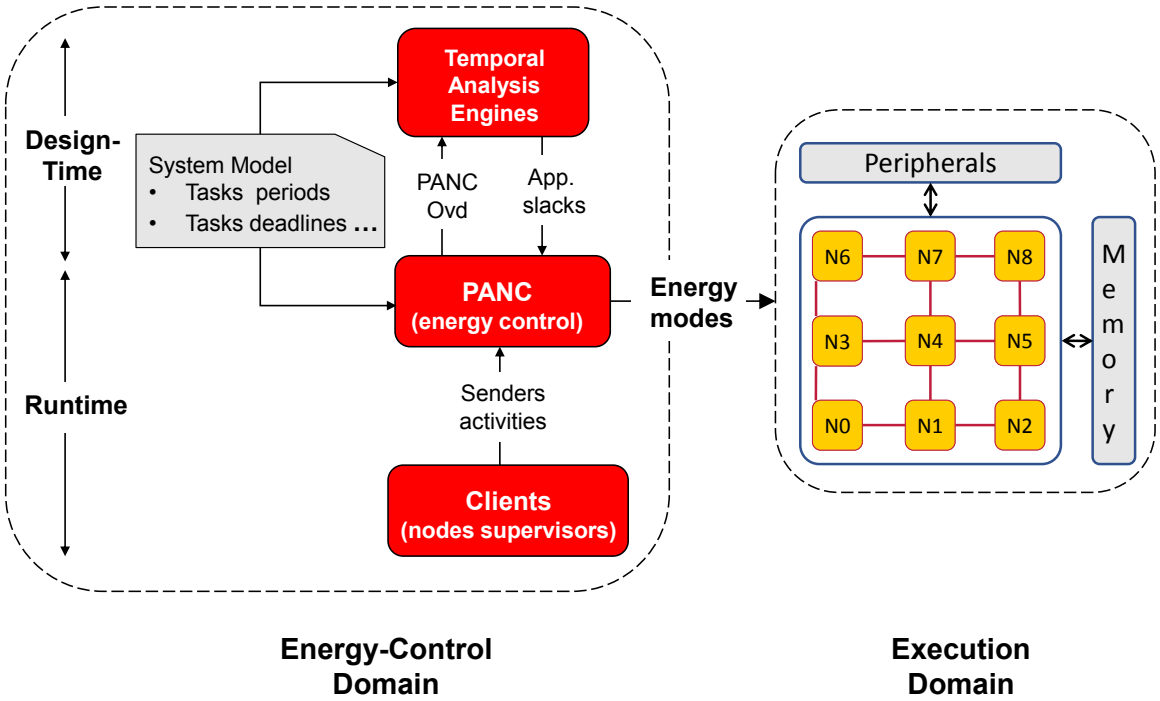


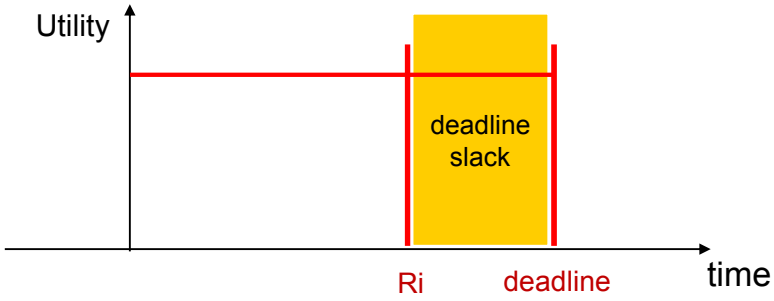
Fig. 2.6.: Block diagram of the comprehensive framework for NoC energy management, employing PANC

ing PANC. The framework mainly showcase the blocks highlighted in orange (PANC, clients, and temporal analysis engines) as substantial units. The clients provide PANC with the global system state dynamically at runtime including the concurrent active tasks. The application tasks are modelled as a set of properties, provided first to PANC as an input at design-time. This contains the tasks routing paths, priorities, deadlines, and NoC different frequencies required to transmit traffic of every set of tasks. Moreover, tasks deadline slacks are also provided to PANC at design-time through the temporal analysis engines/frameworks, which as well rely on PANC latency overhead (PANC_Ovd) to compute the slacks.

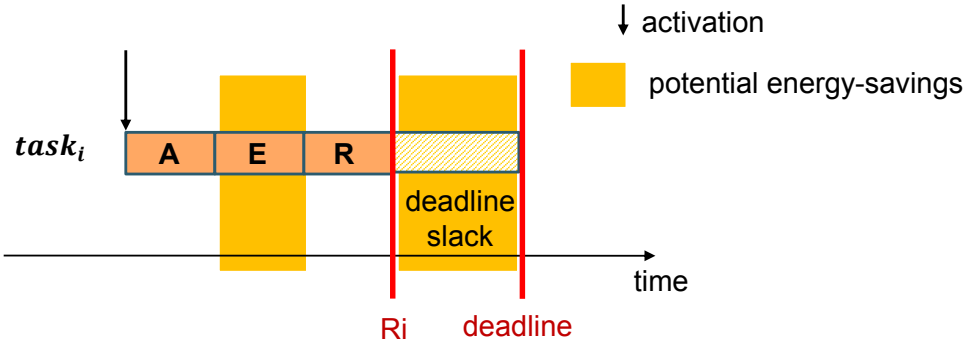
Definition 1. *The task's deadline slack* $(DSlack_i)$ *defines the time budget between the task's deadline* D_i *and its worst-case response time* R_i *accounting for the PANC latency overhead. It is computed as follows:*

$$DSlack_i = D_i - R_i \tag{2.1}$$

In real-time systems, each critical task is characterized by its relative deadline upon which a task should be fulfilled, and thus D_i is a given property [148, 106]. The computation of the PANC latency overhead is performed through the temporal analysis frameworks as detailed later in Section 4.1. Finishing a critical task too early (before its deadline) has the same utility of completing it by its deadline [148].



(a) Utility during a firm/hard real-time task lifetime



(b) Potential energy-savings during the task's execution time (E) and deadline slack phases

Fig. 2.7.: Utility function (a) and potential energy-savings phases (b) during a hard real-time task lifetime

Figure 2.7a illustrates the identical utility of the produced results of a hard real-time task before reaching its firm deadline. Thus, based on the following metrics, (i) tasks models, (ii) global system state, (iii) current routers energy modes, and (iv) the deadline slacks, PANC saves energy by enforcing new energy modes for the NoC routers.

As we conform our approach to hard real-time systems where tasks deadlines must be met, PANC exploits the positive deadline slacks to leverage the energy-savings schemes (power-gating, clock-gating, and *DVFS*). Overall, PANC utilizes three existing factors to save NoC energy on their basis:

- ***Task's Deadline Slack***: See Definition 1.
- ***Hop-Count Slack***: It defines how far powered-off routers are from the source tile in terms of hops, where a hop implies link and router. PANC exploits the Hop-Count slack of the powered-off routers to mitigate the additional latency the packet may experience along its path due to the powered-off routers. This factor is detailed in Section 3.3.1.
- ***AER task-model***: It is a predictable task accesses model, which decouples task's execution phase from communication one [50]. The AER defines a predictable access pattern of a safety-critical task by decomposing it into three consecutive

phases: acquisition (read), execution, and restitution (write). The AER model is a typical model in embedded systems, especially for real-time and safety-critical domains. The task-model not only increases the potential predictability [50], but also the energy-savings using PANC as a task needs to synchronise its NoC access once for the entire transmission (burst of packets). That allows to limit the number of synchronisation messages as well as save energy exploiting the task's execution time to reduce energy/turn off routers. Figure 2.7b depicts the potential phases during a task lifetime (task's execution time and its deadline slack), under which PANCs investigate the potential energy-savings of real-time NoCs.

We detail in the following the protocol-based synchronisation with the control layer that supports PANC with the system global state, essentially required to optimize NoC energy.

2.2.4. Synchronisation Protocol

The power-aware network controller essentially relies on the global state of the system to optimize the NoC energy. The global state corresponds to the cores (concurrent active tasks defining cores requirements for accessing NoC), besides the global state of the network (energy settings, e.g., turned on/off routers and the frequency scale). While PANC is autonomously aware of the latter, the former must be supported by local monitors, the clients. This section illustrates a protocol-based synchronization that is thoroughly conducted between the clients and PANC.

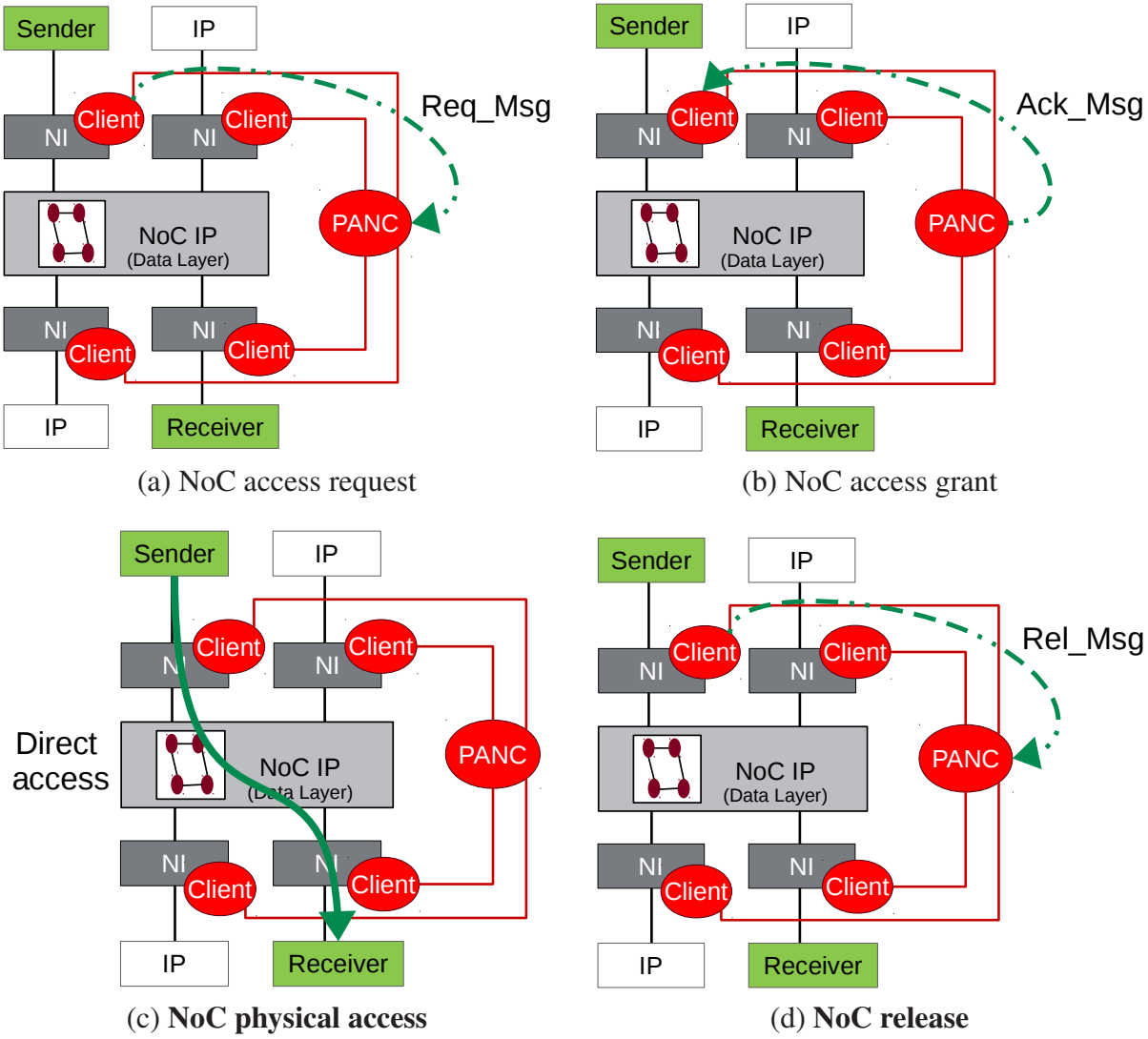
Overall, the global synchronization can be divided into four phases: request, configuration, grant, and release. The synchronization process is exemplified in Figure 2.8. First, the application's task, representing a sender, initiates a request to the client for providing access to, e.g., a remote tile through the NoC. Next, the client, in turn, sends the request message *Req_Msg* to PANC and blocks the sender until PANC allows the communication (Figure 2.8a). After that, PANC processes the request that might require an update of the NoC energy configuration. Based on the NoC energy mode, system model and timing parameters provided by the analysis engines (cf. Figure 2.6), PANC grants the sender via issuing an acknowledgement message *Ack_Msg* (Figure 2.8b). Upon the *Ack_Msg* delivery from PANC, the client unblocks the sender and allows the NoC access (Figure 2.8c). The sender accesses the NoC for the length of the whole transmission (i.e., burst of packets). Eventually, after the transmission completes, the client concludes the communication and releases the NoC via issuing a release message *Rel_Msg* to

Table 2.1.: Overview of the control messages

Message Type	Message	Description
Control Messages	*Req_Msg*	NoC access request message from an active sender to PANC, including the destination address
	Rel_Msg	release message from the destination to PANC
	Stp_Msg	stop message from PANC to active senders to stop sending packets in order to safely reconfigure NoC
	Ack_Msg	NoC access acknowledgement message from PANC to the sender/ an Ack from the stopped sender to PANC
	Rsm_Msg	resume message from PANC to the stopped senders

PANC (Figure 2.8d). After PANC has processed the Release (Rel) request, the NoC energy modes are considered to be adjusted. Note that Table 2.1 presents an overview of the different various of the control messages employed by the protocol.

Now we discuss the synchronisation protocol more in detail. The message sequence chart in Figure 2.9 exemplifies the different energy modes enforced on NoC by PANC to dynamically adapt to current system state requirements for NoC accesses (dynamic task behaviors). The example depicts the concurrent NoC accesses by two tasks. Task_1 starts first and issues a Request (Req) to NoC access via client_1. At this moment, PANC does not require to update the current NoC energy mode and thus it grants the Req via an Ack. Task_1 starts using the NoC and simultaneously task_2 issues a Req to NoC access via client_2. Upon the delivery of the Req from client_2, and precessing it, PANC decides to change the NoC energy mode to adopt the new task transmission. Thus, it first stops the transmission of task_1 via sending a stop message *Stp_Msg* to client_1 and then update the current NoC configurations. The new configuration settings are deployed after assuring the proper transmission of the initiated packets through NoC and that also relies on the actual energy-savings schemes employed by PANC, as detailed later in Section3.3. Once the new energy mode (energy mode_2) is established, PANC grants task_2 and also resumes task_1 via issuing the resume message *Rsm_Msg*.



(a) NoC access request

(b) NoC access grant

(c) **NoC physical access**

(d) **NoC release**

Fig. 2.8.: Synchronisation messages between the clients and PANC via the control layer interconnect

The synchronization workflow for NoC accesses is presented employing one PANC. In the following, we introduce the potential scalability of the control layer to host much larger NoC-based many-core systems, defining the required extensions of the NoC architecture.

2.2.5. Scalability

In safety-critical domains, problems are approached with some sort of centralized controller, however, in many-core systems, with which we concern, this is not scalable. Hence, this section presents the extension of the control layer to better illustrate the proposed mechanism with much larger NoCs. To this end, we first split a NoC into subsets called regions, then assign one PANC to a dedicated

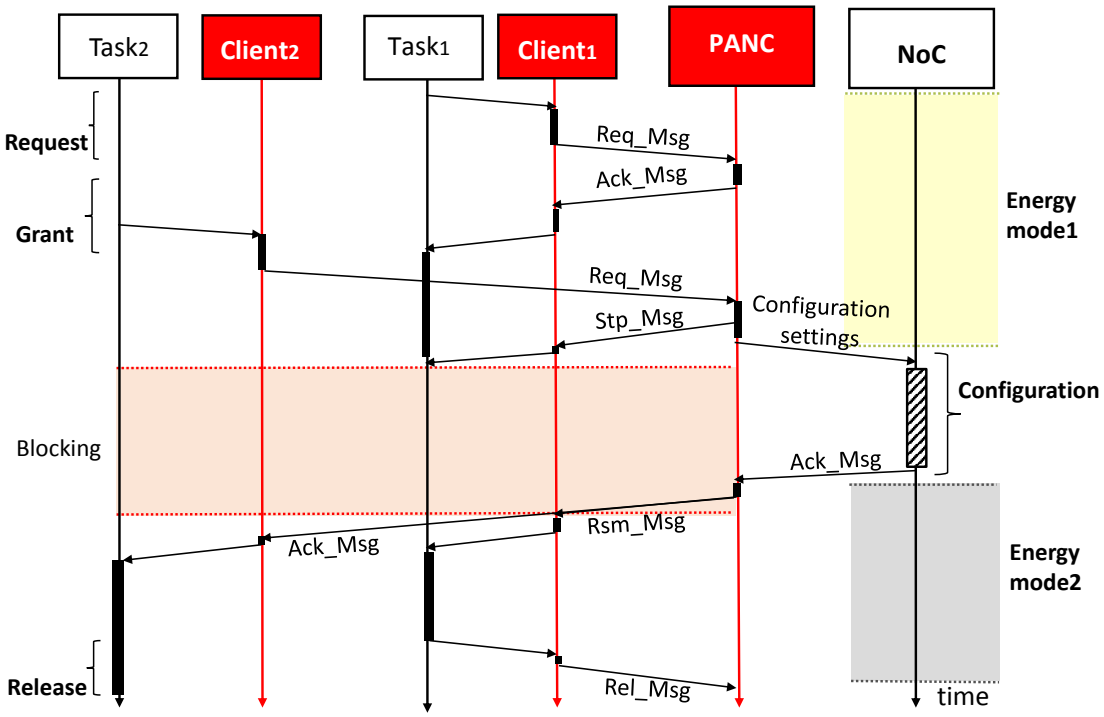


Fig. 2.9.: The exemplar for the synchronisation workflow, highlighting the synchronisation potential phases and NoC various energy modes, initiated due to concurrent active tasks

region. The PANC represent local controllers, and the number of required PANC is basically equal to the number of the NoC regions. Recall, in case of disjoint real-time applications, we could have independent NoC regions [2], and therefore multiple independent PANC where each of them takes care of one NoC region. However, in case of applications comprise inter-region communications, PANC must communicate with each other using an interconnect in order to exchange each other regions' energy states. Thus, we illustrate the architecture of the interconnection between PANC using, e.g., 8×8 NoC then extend it targeting larger NoCs.

The Control Layer under Baseline NoCs

Figure 2.10 depicts the control layer architecture with multiple PANC using baseline NoC, e.g., 8×8. The NoC is split as an example into 4 different regions, each of them is controlled by one PANC (4 PANC together in the NoC). PANC are connected using an additional control NoC, which is only transferring control messages between PANC. The control NoC, in case of 8×8 NoC, is composed of only one switch.

The communication policy between PANC is presented in Figure 2.10. That is, R_{15} (router$_{15}$) in region$_3$, connected by example to an Ethernet port, may

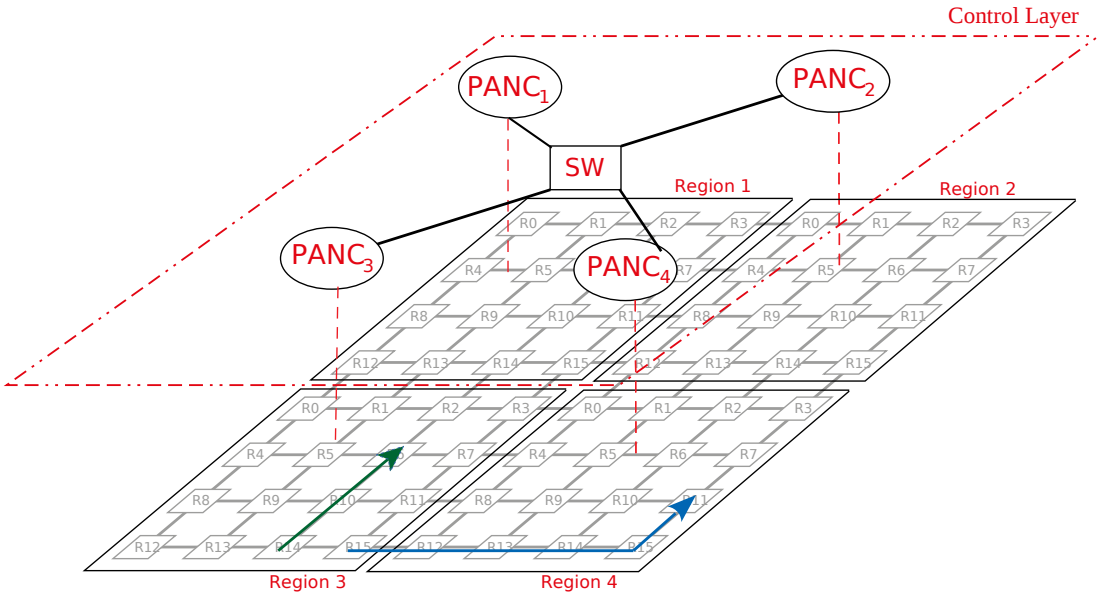


Fig. 2.10.: The control layer architecture employing 8×8 NoC

sporadically send data to R_{11} in region$_4$, connected by example to a memory. Overall, once PANC$_3$ receives a request message from a task in its local region, it checks whether the active task is a local or remote based on its number (considering that the task includes its number in the request message). In other words, it checks in its database whether the task is labelled as local (same region destination), or remote (remote region destination). In case of remote access, PANC$_3$ forwards the request to a remote PANC (PANC$_4$ in our example) located in the corresponding region (region$_4$). The request contains the corresponding source and destination of the packet in region$_4$. PANC$_4$ in turn looks for the current request in its database and reveals the corresponding path. Then, it checks the energy states of the routers in its region, and sends an acknowledgement message *Ack_Msg* back to PANC$_3$ once the routers are ready to conduct the requested transmission. Once PANC$_3$ tackles the local routers, and receives an *Ack_Msg* from PANC$_4$, it acknowledges the request. Note that the processing of routers energy states in each region is based on the energy management schemes employed by PANCs. For example, if PANCs target *DVFS* or its joint application with others schemes (*PG* and *CG*), additional small extensions of the presented architecture are required to deal with different frequency domains through different regions, as described later in Section 4.1.

In order to better clarify the request's content (forwarded from PANC$_3$ to PANC$_4$), Table 2.2 addresses a local and remote tasks. While, in case of a local task, PANC$_3$ acknowledges the request only based on the local routers' states, it acknowledges the request based on the local and remote routers in different regions in case of a remote task. For instance, $task_{15}$ adopts the path from R_{15} in region$_3$ to R_{11} in region$_4$, and it is labelled as remote. Thus, PANC$_3$ tackles the

Table 2.2.: Local and Remote tasks example

Task_Number	Corresponding Path
$task_2\ (local)$	R_{14}, R_{10}, R_6
$task_{15}\ (remote)$	$Reg_3 : R_{15}$, $Reg_4 : R_{12}, R_{11}$

routers in its region (R_{15} in our example), and sends a request to $PANC_4$ containing the corresponding source and destination in $region_4$ (R_{12}, R_{11}). Recall, $PANC_3$ acknowledges $task_{15}$ only after it receives an *Ack_Msg* from $PANC_4$.

Moreover, the link width of the control network (which connects PANC) highly depends on the content of the control message forwarded from one PANC to another. The control message contains the *request/release* signal (required 1 bit), the source and destination of the path in the remote region. As previously mentioned, PANC could run on top of different NoC architectures. However, we assume 2D mesh NoC and XY routing algorithm. Each router demands 4 bits to be addressed in, e.g., 4×4 NoC, comprising 9-bit wide link for addressing the remote path (source and destination) and the request type (*request/release*). Moreover, the request also involves the destination address, i.e., the PANC number, which demands 2 bits in case of 4 PANC in 8×8 NoC. Thus, the link width of the control NoC in 8×8 NoC is 11 bits. Note that we transmit control data in single cycle to speed up the communication between PANCs – mitigating the latency overhead of the control messages and satisfying *Obj1* (cf. Section 2.2.1). Regarding the switch in the control NoC, it is a very simple switch model. It is composed of 4 ports connecting 4 PANC, input buffered with 1VC, and a round robin scheduling policy.

Extending the Control Layer for Larger NoCs

Figure 2.11 depicts the extension of the control layer architecture targeting, e.g., 16×16 NoC. In order to formulate analytical expressions for the proposed extension, we define the following denotations:

- N_{region}: denotes the number of NoC regions;
- N_{PANC}: denotes the number of required PANC;
- NoC_{size}: denotes the size of the control NoC in the control layer;
- N_{SW}: denotes the number of switches in the control NoC;

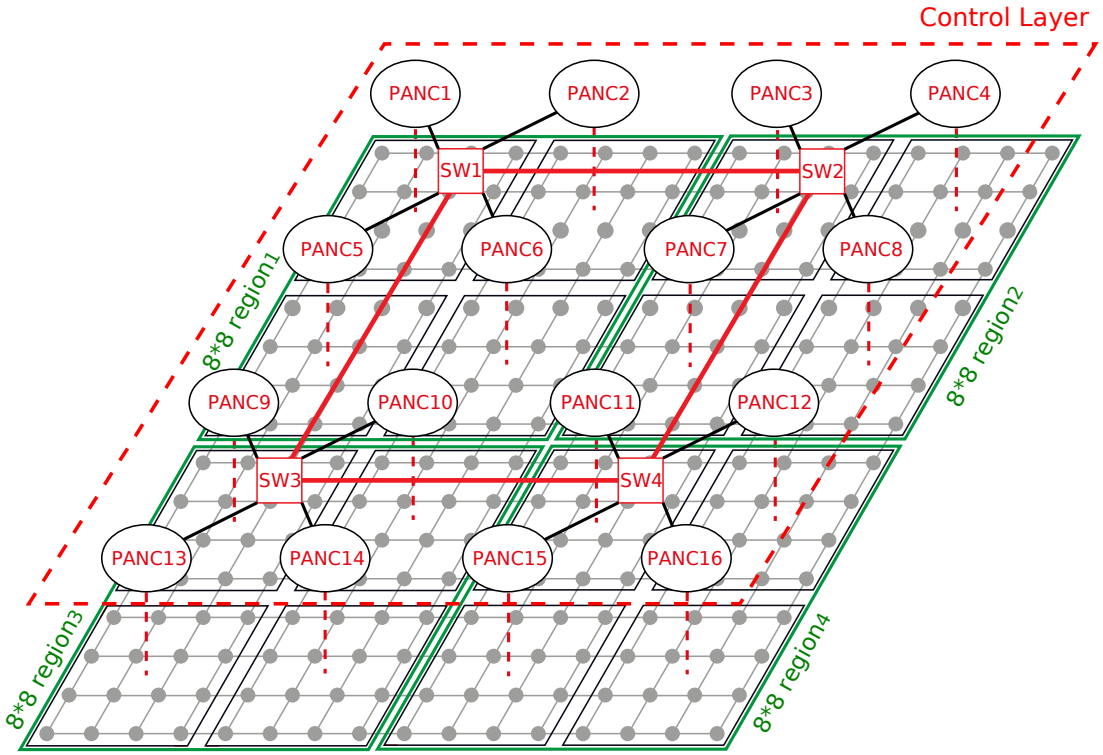


Fig. 2.11.: The control layer architecture employing 16×16 NoC

- L_W: denotes the link width of the control NoC.

First, we derive the number of regions in larger NoC as follows:

$$N_{region} = \text{NoC}_{S\times S} \cdot N_{region:S\times S}, \tag{2.2}$$

where $NoC_{S\times S}$ denotes the number of smaller NoCs comprises the larger one; and $N_{region:S\times S}$ denotes the number of regions of each smaller NoC. We assume in our analysis symmetric square NoCs, and thus the larger NoC comprises 4 times the right smaller one. For instance, the number of regions of $\text{NoC}_{16\times 16}$ is 4 times the number of regions of $NoC_{8\times 8}$. That is, the number of regions of 16×16 NoC is:

$$\begin{aligned} N_{region:16\times 16} &= 4 \cdot N_{region:8\times 8} \\ &= 4 \cdot 4 \\ &= 16 \end{aligned} \tag{2.3}$$

Moreover, as we dedicate one PANC to one NoC region, the number of PANC is always equal to the number of regions:

$$N_{PANC} = N_{region}. \tag{2.4}$$

Regarding control NoC, the size of the NoC is determined as follows:

$$NoC_{size} = \frac{N_{SW}}{2} \times \frac{N_{SW}}{2}, \tag{2.5}$$

where the number of switches (N_{SW}) is subject to the PANC' number, and the number of switches in the smaller NoC ($N_{SW:S\times S}$) as follows:

$$N_{SW} = \frac{N_{PANC}}{N_{PANC:S\times S}} \cdot N_{SW:S\times S}, \tag{2.6}$$

where $N_{PANC:S\times S}$ denotes the number of PANC in smaller NoC. Thus, employing the aforementioned symmetry, the number of switches in 16×16 NoC is:

$$\begin{aligned} N_{SW} &= \frac{4 \cdot N_{PANC:8\times 8}}{N_{PANC:8\times 8}} \cdot N_{SW:8\times 8} \\ &= 4 \cdot N_{SW:8\times 8} \\ &= 4 \cdot 1 \\ &= 4 \end{aligned} \tag{2.7}$$

Regarding the link width of the control layer, it can be calculated using the following equation:

$$L_W = L_{Path} + L_D \tag{2.8}$$

$$L_D = \lceil log_2(N_{PANC}) \rceil, \tag{2.9}$$

where L_{Path} denotes the number of bits required to encode the remote path in the request type; and L_D denotes the number of bits required to encode the destination (the PANC). We remind that L_{Path} is 9-bit wide, and thus, in case of 16×16, the link width is:

$$L_W = 9 + \lceil log_2(16) \rceil = 13 \; bits. \tag{2.10}$$

Based on the aforementioned analysis, Table 2.3 demonstrates the control layer infrastructure under different NoC sizes.

Moreover, Figure 2.12 depicts the PANC and switches increase under different baseline NoC sizes. As shown, PANC increase exponentially to the size of the baseline NoC. However, with larger NoC sizes, e.g. 32×32, the increase becomes linear. That is mainly attributed to the relative increase of the NoC regions. In other words, the absolute increase of the NoC regions from 16×16 to 32×32 is 48 regions (Inc=48), which, obviously, demands larger number of PANC. That way, the larger the NoC is, the conservative energy-savings are achieved employing the corresponding PANC.

Table 2.3.: The control layer infrastructure for different NoC sizes

Baseline NoC Size	N_{PANC}	N_{SW}	NoC_{size}	L_W (bits)
8×8	4	1	1×1	11
16×16	16	4	2×2	13
32×32	64	16	4×4	15

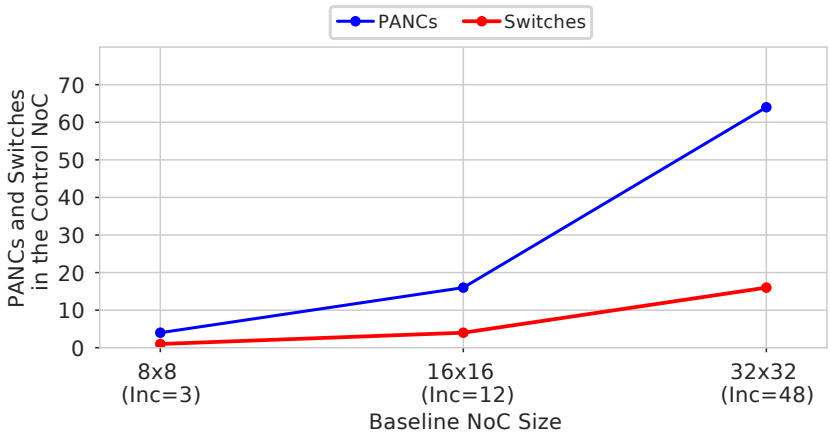


Fig. 2.12.: PANC scalability under different baseline NoC sizes

Overall, employing an additional physical network to host the communication between PANCs is reasonable. We modify the NoC architecture supporting the control layer, as highlighted in Figure 2.5, to present the required changes in the architectural design to host much larger NoCs. Figure 2.13 illustrates the integration of the scalable control layer in the baseline NoC. In this case, the transport layer is extended to include not only PANCs with the direct links but also the control network, connecting PANCs with each other. Moreover, the newer generation of MPSoC is almost always heterogeneously structured with a heterogeneous NoC. This work employs homogeneous NoCs, however, supports as well heterogeneous NoCs as the control layer is isolated from the underlying data transmission network layer and thus it is transparent to the routers' architecture.

In the following, we present an extension of the NoC energy control concept to include system-level degradation treatment through introducing hierarchical control layers that cooperate with each other to achieve this goal.

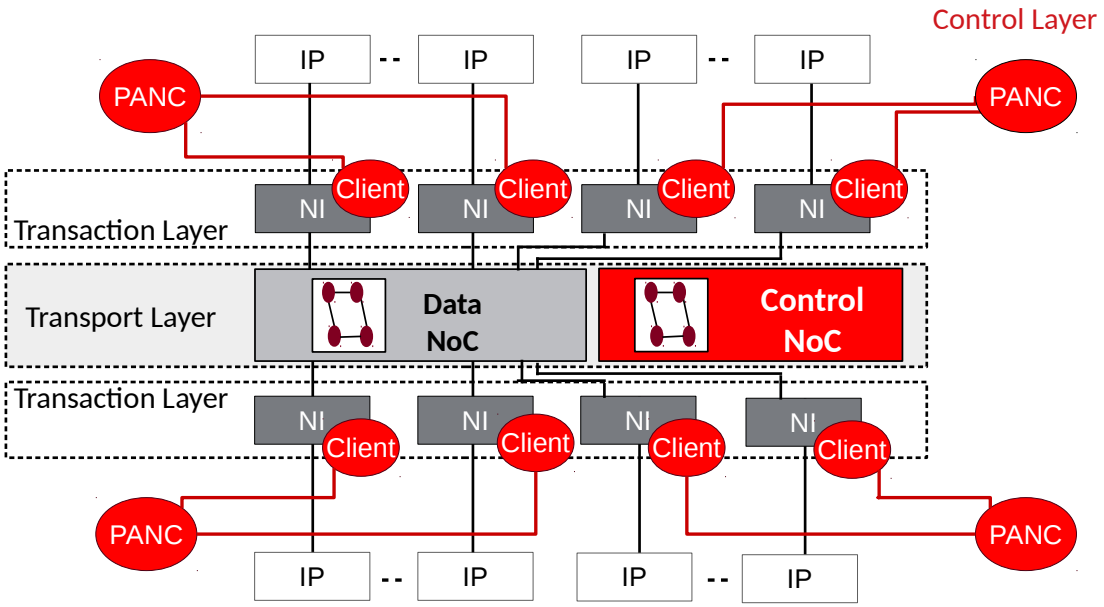


Fig. 2.13.: The high level view of the control layer integration (red color) under scalability

2.3. System-Level Hierarchical Control Integration

Multiprocessor architectures are emerging platforms used in mixed-criticality systems to provide high performance and tight energy requirements. A resulting challenge is the control, optimization, and reliable operation of such complex multiprocessing architectures. These systems must manage the resource usage at runtime, adapting to system changes due to, e.g., degradation.

To solve this, we develop an online management approach to address the response to system degradation. The approach corresponds to proactive, hierarchical control layers which, in synergy, manage and configure such large systems. A variety of mechanisms is employed including proactive reconfiguration to mitigate the risk of failures, and chip-level operation planning with flexible boundaries between safety-critical and best-effort subsystems. Being proactive enables an early reaction to cursors of degradation, resulting in high responsiveness of the system to imminent hazards before they manifest a serious risk. In this work, we focus on the treatment of imminent hazards that might occur due to degradation (e.g., imminent core failure). An imminent hazard is defined as follows [131]:

Definition 2. *Imminent hazard: an increased risk of future errors that can lead to system failure and, therewith, a hazard.*

The online management approach with a proactive nature provides an infrastructure for system introspection and reflective behavior, which is the foundation

for computational self-awareness in mixed-criticality systems. Computational self-awareness is the ability of a computing system to recognize its own state, possible actions and the result of these actions on itself, its operational goals, and its environment, thereby empowering the system to become autonomous [83]. The approach corresponds to system reconfiguration that adapts the system to the current change. The reconfiguration involves container migration, which migrates the endangered safety-critical workload to a resource with reduced risk.

Definition 3. *A container encapsulates a software stack required for the execution of the respective workload, including runtime environment, and an operating system.*

In the following, we present a concise overview of the system architectural design supporting hierarchical and distributed managers to provide dependable operation in case of system degradation that results in imminent failures.

2.3.1. Architectural Design

The mixed-criticality system architecture is illustrated in Figure 2.14. The hardware architecture, depicted in the lower part of Figure 2.14, consists of many tiles that connect with each other via a shared main memory through a NoC. NIs connect the tiles to the NoC. Tile contents vary from processing elements (PEs) and SRAMs to memory controllers and I/O interfaces. The components of a tile, e.g., processor, timer (tmr), and generic interrupt controller (GIC) are internally connected to each other and to the NI by a local bus.

The platform must provide sufficient independence between applications with different criticalities in order to support mixed-criticality systems [78, 81, 49]. These requirements are satisfied by spatial isolation between tiles and bounded interference in shared resources. Spatial isolation is achieved with the tile-based architecture together with a NoC for mixed-critical real-time systems [153]. The tiles are organized in two zones, Safety-Critical (SC) zone and BE zone. The SC zone is hosting safety-critical applications, and the BE zone is hosting non-critical applications. Each PE, regardless of it belonging to SC or BE, is a uniprocessor with fully preemptive fixed-priority scheduling. The workload executed on each PE consists of application's tasks, and eventual migration-related actions, where the actions are always assigned higher priorities than the tasks.

The software architecture, depicted in the upper part of Figure 2.14, consists mainly of containers (see Definition 3). The mixed-critical workload is partitioned into BE containers and SC containers, which are respectively mapped to BE and

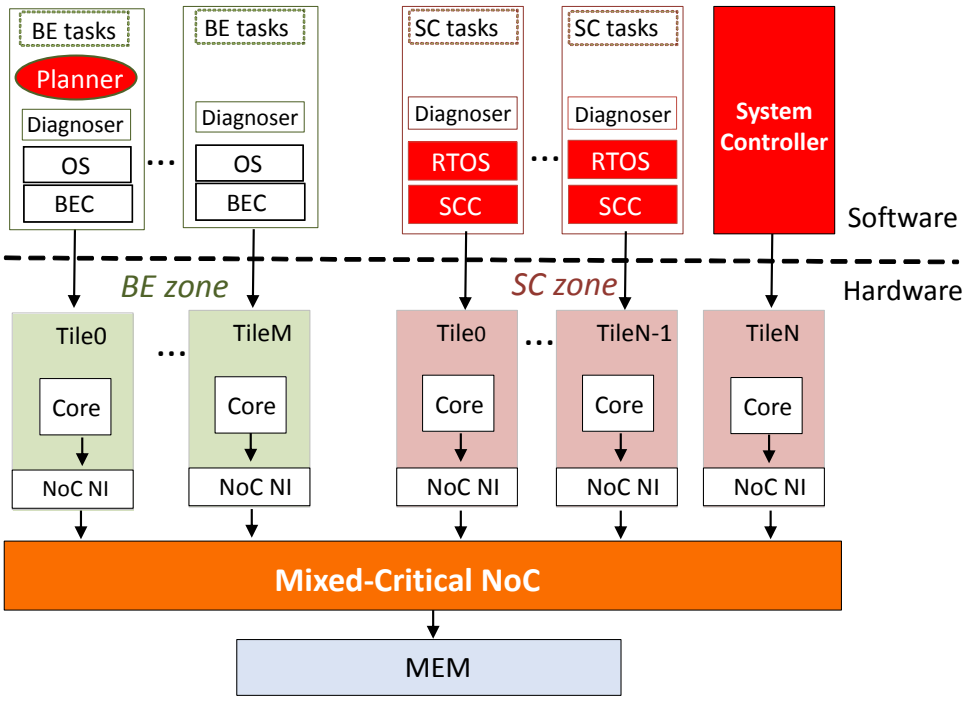


Fig. 2.14.: Architectural template of NoC-based many-core mixed-criticality system, supporting dependable operation through local and global managers

SC processing tiles. The SC zone runs two kinds of entities, the system controller (SyC) and the SC containers. The system controller has the ultimate control of the platform through features of the underlying software and hardware architecture. Moreover, the container must be tailored for its workload. That is, the SC container includes a Real-Time Operating System (RTOS) for temporal guarantees and predictability. A Critical Tile Controller (CTC), which is a SC tile controller, constitutes an interface between the system controller and the RTOS. The CTC reacts on the system controller indications on time. A diagnoser, representing an on-tile self-diagnosis component, operates on both BE and SC zones. It is responsible for monitoring the system for errors and imminent hazards [142].

In turn, BE containers include a software stack that include an Operating System (OS), and also a Best-Effort Controller (BEC) that manages and controls the best-effort workload [129, 132]. The BEC also accounts for the system controller indications and reacts based on it. The planner is responsible for long-term planning of the system. It develops alternative configurations (i.e., reconfiguration of containers-to-tiles mapping) required for handling imminent hazards/failures. The plans are developed not only at design time but also at runtime upon system changes. The planner is not timing-critical as it plans for future hazards that have not been manifested yet and thus it resides on a BE tile.

Note that the imminent hazard detection that is supported by the diagnosers [105, 143], and managing the BE workload by the BE controller [129] are orthogonal

to this work that mainly focuses on the functionality of the SC part of the system architecture (highlighted in red), which are briefly described in the following of this chapter.

System controller and RTOS

The system controller is responsible for enforcing safe system configurations by globally monitoring, and controlling the layers below, under the guidance of the planning entity. Changes to the system configuration are realized with transitions from a Current Operating Point (COP) to a Next Operating Point (NOP).

Definition 4. *An operating point (OP) is a specific configuration for the containers, resources, containers' mapping and resources' allocation.*

Small changes in the OP, e.g., changing the resources configuration keep the OP within its safety margins under which the system is still safe [138], however, going beyond controllable margins requires transitioning to a NOP to bring back the safety margins [132, 135].

The CTC extends the RTOS to monitor and control the safety-critical part of the system (the SC zone), according to the system controller requests. The System Controller (SyC) has an ultimate control over the entire system coordinating the control of BE and SC zones respectively with BEC and CTC. It is also responsible for configuring the system shared resources according to the current system state [86]. The SyC monitors the system for changes/events in the environmental or operating conditions that impact the execution of the SC workload. Events refer to, e.g., a change (software update), errors, or to a predicted change that will be proactively handled, such as an imminent failure of a processing resource (with which we concern).

Transitions from a COP to a NOP are carried out by the SyC with the collaboration of local controllers, BEC and CTC. Such transitions can include changes to the configuration of a single resource or can include a complete change in the mapping of containers to resources. Shared resources like NoCs are also reconfigured by the SyC since they must comply with the highest levels of criticality [81]. Note that the time to transition between system states varies depending on the amount of reconfiguration involved. During a transition, resources can be added or removed from the SC and BE zones. When the SC and BE zones are resized, i.e., resources are added to or removed from a zone, a controller appropriately releases its resources before handing them over to the other zone. Also, the transition of a resource from BE zone to SC zone is timing-critical. Therefore, the SyC is allowed to forcefully execute that transition without BEC, in case the latter takes too

long to release it, in order to make the execution of the SC workload independent of the execution of the BE workload (sufficient independence) [81, 78, 49], which is detailed later in Chapter 5.

Planner

It plans future proactive actions, taking into account the operating conditions of the system, error rates, energy consumption, aging, temporal constraints, and other events that may occur at runtime. The main responsibility of the planning phase is to create and maintain the set of plans that are used by the system controller to handle imminent hazards.

Definition 5. *The imminent hazard together with the corresponding configuration (including the BE tiles to which we can migrate the SC container) are called a plan.*

The planner only includes a new plan in the set of plans if it meets all non-functional requirements of the SC workload. That includes system-level performance analysis tools such as compositional performance analysis (CPA) [68, 155]. Besides, transitions involve the remapping of containers to resources. Remapping requires moving code and data and therefore impacts the response time of the executing workload, which can lead to a system-level timing violation (deadline misses). Thus, the planner must also check for system-level timing and safety violations of a transition before considering it as a future plan.

Note that the functional details of the hierarchical control approach is introduced in Chapter 5. That is, we present the communication protocols between local and global controllers to perform container migration adapting to system severe changes (e.g., imminent core failure), while preserving fundamental requirements in mixed-criticality systems.

2.4. Summary

To cope with the ever-increasing complexity of interconnected functions, and to reduce the cost and energy consumption of a system, multi- and many-core systems are utilized to efficiently integrate multiple processing elements on a single chip. In turn, these systems must hardly manage the resource usage to adapt to system changes satisfying safety targets. This chapter has introduced the main principles of NoC-level energy optimization and system-level degradation treatment.

Networks-on-chip, as scalable and modular interconnects, are employed in MP-SoCs. The associated high energy consumption is challenging and can lead to system reliability issues. The NoC control layer has been proposed to dynamically manage and control NoC energy consumption. For mixed-criticality systems where safety-critical hard real-time applications are integrated with best-effort applications, hard real-time NoCs are employed. That is, NoCs are supported with mechanisms that account for real-time requirements of timing-critical traffic [152, 97]. Consequently, NoC energy management must preserve temporal predictability. The energy control layer comprises PANCs and on-tile local supervisors, the clients. PANCs safely save energy on NoC routers, based on their global knowledge of the system state, mainly provided by the clients. PANCs exploit the AER task model to save energy and apply energy-savings schemes like (*PG*, *CG*, *DVFS*) depending on the applications deadline slacks. The latter define how much time is available to reach the deadline after accounting for the worst-case latency induced by the control layer. Thus, the NoC control layer optimizes NoC energy under timing constraints.

Moreover, the problem gets more complex when other situations like errors or imminent failures are considered. This chapter briefly presented the hierarchical and distributed local and global controllers that enforce system reconfiguration. The controllers are organised into hierarchical control layers that extend the scope from NoC energy management to system-level degradation treatment. The layers enable reflective systems where concrete cooperation between local and global controllers is conducted to reconfigure the system and resolve the respective risk. Long-time planning has also been proposed to provide configuration setups targeting potential future changes that will be handled by the controllers. The proposed runtime management must account for safety requirements [81, 49, 78], while transitioning the system from the current state to a new one. To detail this, Chapter 5 introduces deterministic communication protocols between local and global controllers to safely adopt system changes, while preserving fundamental isolation requirements for mixed-criticality systems.

Chapter 3: Energy Management for Hard Real-Time NoCs

The increased complexity of MPSoC has also caused an immensely increase of their associated energy consumptions that might affect system performance and reliability. Networks-on-Chip are the prevalent solution to provide a scalable interconnect for such complex multiprocessing architectures [92]. As previously descried, these complex platforms come at the cost of high NoC energy dissipation that requires thorough management. Energy dissipation is mainly caused by dynamic and leakage energy sources. Dynamic energy itself corresponds to clock-tree and switching energies. Indeed, a significant portion of NoC energy consumption is induced by the static power [38]. In addition, static power is anticipated to be exacerbated in future transistor size. On the other hand, the authors in [13, 89, 115] state that the clock-tree power is a key contributor to total NoC power. As up to 81.3% of a router quiescent power (i.e. no load) is caused by the activity on the clock pins of high-level non-clock gated synchronous elements [89], as introduced later in Section 3.2. That in turn constitutes a huge energy loss in idle cases. Thus, energy management mechanisms for NoCs have to be developed alleviating various sources of energy dissipation.

This chapter first demonstrates the breakdown of the power dissipations of hard real-time NoC routers designed for Application-Specific Integrated Circuit (ASIC) to investigate accurate power figures (Section 3.2). Next, it introduces a global energy management control layer to address the energy loss of hard real-time NoC routers, preventing thermal runaway and reliability issues (Section 3.3). As introduced earlier, the control layer approach dynamically manages the energy consumption in NoC based on the workload dynamic behavior, while assuring real-time constraints of timing-critical applications. To this end, the control layer fully exploits the application properties (e.g., the slacks) through its primary units, the PANCs.

We explore, through PANCs, the potential efficiency of integrating multiple energy-savings schemes in the face of the diversity of energy dissipation sources. That is, PANCs explore applying one of the following schemes: *PG*, *CG*, *DVFS*,

or integrate them in one platform so that PANCs investigate the joint application of *PG* and *CG*; and of *PG*, *CG*, and *DVFS* targeting both dynamic and leakage energy optimizations, resulting in energy-aware NoCs The control layer is developed on top of the existing NoC data layer. This decouples the mechanisms responsible for energy-savings (PANCs) from the underlying NoC infrastructure conducting switch arbitration between packets (the data layer). The substantial motivation beyond the isolation between NoC data and control layers is to provide the users flexibilities of employing our energy management approach without introducing complex and hard to maintain modifications to their already existing NoCs. This is, by the way, the same policy employed by some commercial available NoCs, e.g., 256-core in MPPA from Kalray [25, 46] and tile64 from Tilera [151, 16].

The rest of this chapter is organised as follows. Section 3.1 reviews the relevant related work concerning the energy management schemes of NoCs. Section 3.2 introduces the power analysis framework and demonstrates the power figures of NoC routers designed for ASIC. Section 3.3 presents the individual/joint application of diverse energy management schemes (*PG*, *CG*, and *DVFS*) to allow efficient energy-savings in hard real-time NoCs. Finally, Section 3.4 concludes the chapter.

3.1. Related Work

As motivated, NoC's power consumption constitutes a significant ratio of total chip power, which in turn requires intensive efforts to tackle the respective energy loss. Several NoC energy management schemes have been intensively researched, where energy-gating has been successfully applied. However, applying them to hard real-time and safety-critical systems makes critical functions vulnerable, and thus jeopardizing temporal guarantees. Hence, resolving the trade-off between energy-savings granularities, while keeping the system predictable is crucial. The diversity of sources of energy loss calls for an integration of multiple energy-savings schemes. This chapter discusses the impacts of different energy-saving schemes, including Power-Gating, Clock-Gating, and Dynamic Voltage and Frequency Scaling, and reviews their usage by the state-of-the-art.

3.1.1. Power-Gating

Due to technology downscaling and relative reduction in supply voltage, the leakage power has been increased and constituted a significant proportion of the chips total power [58]. Power-Gating corresponds to turning NoC routers on/off

to feature leakage (static) power reduction. *PG* schemes have been intensively researched and applied on NoC systems [74, 125, 39, 58, 57]. Besides NoC routers, *PG* has been applied to cores and execution units [77]. This application of *PG* at core level emphasizes the high impact of this approach on energy-savings.

Bufferless routing [57] saves router power by eliminating buffers, but it might induce issues like livelock, misrouting, and packet reassembly that must be appropriately tackled. Furthermore, either portion of a router buffer is powered off [32], some blocks of a router [32, 125] are powered-off, or the whole router [38, 112, 39, 58]. In addition, path-oriented fine-grained Power-Gating mechanism (PAPM) [55] is also presented. It selectively powers on/off paths on the network, partially shared by different sources. That is, unused queues for congested traffic can be powered off. However, the presented solutions with powering off only part of a router have limited energy-savings capabilities.

Panthre [125] and NoRD [38] are reconfiguration-based NoC power-saving schemes. Panthre is based on the observation that only 10% of network traffic flows through 30% of the links, providing a potential for Power-Gating by detouring packets to exclude lightly used portions of the NoC. However, in addition to area penalty, this method suffers from issues associated with reconfiguration such as detour. Moreover, it increases some routers' load in order to keep others sleep, thus accelerating aging of those high-loaded routers and leading to an age unbalance between routers [63]. NoRD, on the other hand, uses bypass paths to circumvent powered-off routers. It relies on packet detours which has the same aforementioned reconfiguration drawbacks. Furthermore, Catnap [44], uses multiple networks to increase the efficiency of Power-Gating, however, it does not account for temporal requirements.

Overall, the primary challenges of all Power-Gating schemes are when and how the routers could be powered on/off to save energy under negligible latency overhead. Moreover, a Power-Gating itself comes with power overhead, which should be addressed thoughtfully. The latter mainly comes from the power lost in turning routers on/off, i.e., turn on/off the relative power switch, connecting router to power supply. Hence, the respected energy loss should be compensated so that the application of Power-Gating does not adversely increase power dissipation. A variety of research work has been made to fulfill that goal and overcome the Power-Gating challenges. Conventional *PG* schemes turn off routers once they go to idle mode. Matsutani [112] proposes an optimization of the conventional schemes by introducing an early wake-up technique. It is mainly based on look-ahead routing that sends a wake-up signal two hops ahead of the packet arrival. However, the method only sends the control signals at most 2 hops ahead, and in turn hides only a small fraction of the wake-up latency. An optimization to

the look-ahead routing is introduced by the work of Chen [39]. He proposes a power punch technique which sends multi-hop wake-up signal in order to almost hide the overall latency overhead. However, the method still suffers from the fragmented power cycling that reduces the potential energy-savings. On the other hand, the method generates wake-up signals by the network interface to bunch all routers along the packet's path, which might be already on. That in turn requires generating and monitoring many useless power punch signals.

TOOT (Turn-on on Turn) is a mechanism that reduces the total number of wake-ups by providing a bypass path for straight/eject packets [58]. Consequently, TooT improves the efficiency of *PG* mechanism from power perspective, however, at the cost of performance as packets have to wait for each other in Toot's bypass latch.

In general, the previous work suffers from various issues. *First*, none of the aforementioned *PG* schemes provides guarantees for hard real-time constrains as they focus on best-effort systems. *Second*, we do not limit the work to certain usecase observations [58, 125], and estimate the overhead of *PG* mechanism independently from the NoC topology, as opposed to [39, 58]. *Third*, as it can be summarized from the previous work, they either save power efficiently at the cost of performance [58], or provide very light latency overhead but less power-savings [39]. Our aim is to overcome the related work limitations and shortcomings to better saves leakage energy.

Power-Gating schemes are limited to solely leakage energy-savings, however, clock-tree energy dissipation is also significant and should be tackled as described in the following section.

3.1.2. Clock-Gating

Clock-tree power has been addressed by previous work [13, 89, 115] as a major contributor to power consumption as it increases the quiescent power (i.e., no load). Thus, *Clock-Gating* scheme has been applied to gate the clock source and optimize the clock-tree energy dissipation where applicable. Indeed, gating the clock once detecting idle states induces highly potential reduction of the clock-tree power. To this end, automatic *CG* is enabled during synthesis [89, 115]. That means the tool automatically inserts low-level clock-gating cells once suitable enabling conditions are detected. However, as introduced in [89], the activity on the clock pins of high-level non-clock gated synchronous elements still leads to a clock-tree power overhead. To tackle this, the authors in [115] propose gating the clock at router level, and ensure no dissipation is induced by the clock-tree when the router is idle. However, the work does not provide temporal guarantees

for safety-critical hard real-time functions when the NoC power mode changes, i.e., switching from gated to ungated clock. Thus, we aim to support guarantees of timing-critical functions, while saving clock-tree energy dissipation.

As Clock-Gating schemes only feature clock-tree energy reduction, in the following, we discuss the status quo of switching energy reduction schemes.

3.1.3. Dynamic Voltage and Frequency Scaling

Both leakage and switching energy sources have potential reduction employing Dynamic Voltage and Frequency Scaling. It is a very popular scheme that reduces the NoC energy consumption by adjusting the NoC frequency according to its utilization rates. The problem of reducing the NoC energy consumption employing *DVFS* was already addressed by multiple research efforts. However, in the majority of studies (e.g., [75, 161, 40, 163]) the focus is on performance, i.e., best-effort NoCs without hard real-time requirements. In such setups with best-effort applications there are two predominant strategies: (i) to adapt the voltage/frequency of individual routers or links, e.g. [75, 161], and (ii) to adjust the settings for the whole network, e.g. [40, 163].

The contribution w.r.t critical systems with hard real-time constrains was done by Zhan et al. [167, 168]. In these research studies, the authors propose a novel methodology which directly combines the strict latency requirements imposed on NoC with the strategy for energy management. First, the worst-case bound on network latencies are derived and later the deadline slack is estimated (see Definition 1), on which a packet can be further delayed without violating its deadline. Subsequently, the frequency/voltage is reduced to trade slack for decreased energy consumption. Although effective in certain setups, this method provides the energy management scheme which is limited to a static set of interfering senders and optimized w.r.t the worst-case. Consequently, this method has two significant limitations. The worst-case estimation based on the formal analysis can be overly pessimistic leading to marginal improvement resulting from very low slack-levels. Second, in many modern embedded applications, e.g., autonomous driving, truck platooning etc., actions must be taken in the context of a dynamic, constantly changing environment (e.g., situation on the road, quality of sensor data). Consequently, setups are highly dynamic, leading to mode-dependent and data-dependent task's behavior and substantial changes in traffic patterns. Thus, the worst-case scenario (all senders running concurrently and transmitting with maximum frequency) may happen only for a small fraction of time and the average load might be significantly lower.

Although one could extend the method of Zhan et al. [167] to calculate the slack

for different number of active senders, it is necessary to thoughtfully organize transitions between different modes, in order to avoid transient overloads. Note that the presence of overloads could potentially lead to missed deadlines, and consequently hazardous system behavior with severe and catastrophic consequences.

The other approach is to divide the chip into frequency domains, i.e., Voltage-Frequency-Islands (VFI) introduced in, e.g. [95, 82, 124, 94]. VFI from the perspective of the real-time applicability have two serious disadvantages. First, each region suffers from the same lack of dynamics in energy management as in the classic *DVFS*. Second, they are hardly predictable whenever there is an inter-region traffic. Indeed in the majority of NoCs, e.g. Tile64, MPPA, the nodes peripherals (e.g., DDR-SDRAM) are placed at the edge of the chip. Consequently, it is often impossible to avoid inter-region traffic. In case of regions with different frequency, this could cause flooding of the slower region with messages from higher frequency domain - propagation of blocking and backpressure whenever the slower region cannot forward messages as soon as they arrive. In this work, we propose, through PANC, to exploit the deadline slack and introduce dynamic voltage and frequency scaling, i.e., adjust the energy consumption during runtime to the actual load of the NoC.

Although *DVFS* targets energy-savings of both leakage and switching energy sources, it is limited to the circuit physical properties. That is, it cannot lower the frequency/voltage below the transistor limited bounds that are set during the design process.

3.1.4. Integral Solutions

Integrating the aforementioned energy-savings schemes (*PG*, *CG*, and *DVFS*) is highly promising as it targets simultaneously different energy sources. In [147, 110] the authors present the joint application of *CG* and *PG* at circuit level. The scheme of [147] chooses only a subset of flip-flops to be gated selectively. Then, the components performing redundant operations during the clock gated period are determined and power gated. In [110] the authors provide an analysis tool which is able to provide a suitable evaluation of a Register Transfer Level (RTL) design in order to determine the efficiency of *CG* and *PG* integration. In addition, the authors in [60] provide a holistic concept, bringing together *PG* and dynamic frequency scaling schemes for NoCs. Other work presents a simulation framework for power-performance analysis of NoC, integrating accurate *PG* and *DVFS* models [170] and comprising their timing and power overheads. Overall, the integration of the energy-savings schemes, introduced in the literature, either does not focus specifically on NoC, which is a shared resource that must be thoroughly

handled [147, 110, 103, 118], or does not demonstrate the safe evaluation of the proposed integration [60, 170].

To this end, we introduce PANCs through a control layer for NoCs, exploring the safe and efficient integration of energy-savings schemes for hard real-time NoCs. Formal timing analysis is provided in Section 4.1 from which the safe application of PANCs is derived. To the best of our knowledge this is the first work that investigates efficient energy-savings through integrating multiple energy-savings schemes, while simultaneously providing hard real-time temporal guarantees for NoCs deployments in safety-critical domains (e.g., automotive).

In the following, we present the power analysis framework employed to compute the power overhead of real-time NoC routers designed for ASIC, investigating accurate power figures. The framework can easily be generalized from router level to evaluate the power of the complete NoC in different sizes, employing a variety of desirable applications.

3.2. Power Analysis

Networks-on-Chip are presented as an interconnect solution in MPSoCs for both ASIC and Field-Programmable Gate Array (FPGA) designs [67]. The NoC design space is large as there are many design parameters that a NoC designer can vary for a certain architecture [67]. There exists a variety of mechanisms and NoCs that have been compared and investigated in terms of power, area and their resulting reliability [20].

For a baseline router, without special support for fault tolerance, the authors in [159, 166, 14] evaluate the power consumption of different router architectures. The authors in [166] estimate the power consumption on different architectures of a Switch Fabric. The ORION simulator [159] provides a simulation framework for estimating detailed power characteristics of a NoC. In [14], Banerjee explores the breakdown power consumption for various different baseline router designs. The introduced NoCs have been developed for FPGA. Those implementations give a first good estimate for the power overhead of a NoC router. However, the different structures of FPGA and ASIC make these estimates inaccurate for ASIC designs. Power dissipation is increasingly becoming a very important metric in hardware design [159], as it plays a significant role in battery lifetime and temperature runaway [164]. Thus, an accurate power calculation is required. In addition, as future MPSoCs demand for a high circuit efficiency, more accurate ASIC based overhead estimations are needed.

Consequently, we provide in the following accurate evaluations of the power

overhead of real-time NoC routers, developed for ASIC.

3.2.1. Analysis Framework

In order to develop our router in IDAMC platform (Section 2.1.6) for ASIC, we modified the RTL description as follows: **style modifications**, which are required to make the design synthesizable by ASIC tools; and **design optimizations for ASIC**, where we implemented the data buffer (see Figure 2.3a) as a set of registers instead of RAM. We used only logic cells as opposed to memory cells since the latter do not pay off for implementing small buffers. After that, we synthesized the baseline fully connected routers in 4×4 2D mesh NoC by employing a standard ASIC design flow. The fully connected router comprises of 5 ports, with 4 ports connecting to neighboring routers and the fifth one connecting to a tile (e.g., processor, memory). A 65nm CMOS process technology from United Microelectronics Corporation (UMC) with core cell libraries of both high and low threshold voltages in worst case corner (WC, 0.9V, 125°C) was selected. Synthesis for ASIC implementation was carried out by using Synopsys tool chain. All the design units of the router were synthesized using a common compile script. A top-down approach was used for this compilation, while preserving the design hierarchy. After implementing the NoC router design as an ASIC, the overhead metrics regarding area and power of such a router are evaluated. In the sequence, we address the measurement methodology for these metrics.

The area for the standard-cell implementation is defined as the area of the final core in the netlist. Design Compiler (DC) reports the hierarchical area for each design unit of such a router. In Section 3.2.2, using the best-effort synthesis method in DC, we present the area overhead for ASIC routers, and report results of simulations at a clock frequency of 550MHz.

Regarding power, we evaluated the power consumed by the router using Synopsys PrimeTime. The reported total power consumed by the router is 43.6mW. This high power figure results from the clock-tree dynamic power. In our design, because of optimizing the router data buffer for ASIC, where the RAM is implemented as a set of registers, the clock-tree has a higher impact on power dissipation. To minimize this effect, we enabled Clock-Gating in our design [115]. This means the synthesis tool automatically inserted low-level *CG* cells whenever appropriate enabling signals were detected in the source RTL design. With *CG*, the introduced superfluous dynamic power is drastically reduced. This is attributed to the fact that in our router each data buffer stores 3500 bits of data (i.e., 5 VCs, 5 flits per VC, 4 Physical Units (Phits) per flit). From those, at most 35 bits, i.e., 1 Phit can be modified in a given clock cycle. Thus, we only need 35-bit

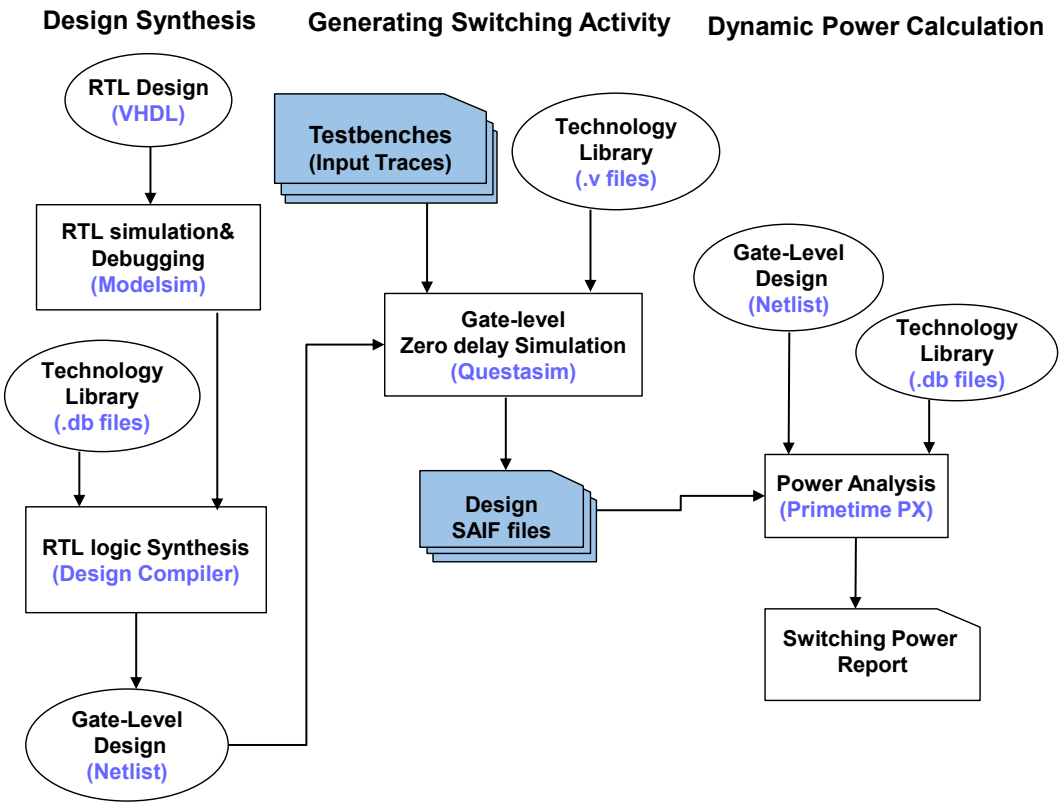


Fig. 3.1.: Design flow for realistic power calculation in ASIC

register per VC to be triggered every clock cycle, and the *CG* provides a strategy to shut down the clock of all other registers when they retain their states [34].

Since the CG design is highly load dependent, we introduce in the following the benchmark-driven workloads to inject at the router inputs and calculate the realistic power, where Figure 3.1 gives an overview of the employed design flow.

Benchmark-Driven Workloads Generation

In order to accurately compute the power consumed by our router (cf. Figure 2.3a), we employ application-driven workloads. Based on that, the switching information is used to gain a reasonably accurate calculation of the power consumption and enable meaningful power optimizations. Therefore, different communication workloads are run and their impact on overall router is investigated. Benchmarks from the MiBench suite [66] are employed as real-time applications generating traffic in the NoC. In our design, we employed two different simulation scenarios to generate two different workloads, detailed in Table 3.1. Moreover, we consider the technique of replicated execution [133], which provides reliable execution of real-time applications on unreliable hardware. To do this, each ap-

plication is mapped twice to our platform (cf. Figure 3.2) in a Dual Modular Redundancy (DMR) configuration [12]. The first scenario is the *Data intensive*, where, on average, each core performs one access (per packet) to the memory every 162 cycles. Since the packet's size is 5 flits and the NoC transmission time per link is 1 FLIT every 6 clock cycles, then the NoC load with the first scenario is 21.15%. The second scenario is Mixed workload, where, on average, each core performs one access to the memory every 502 cycles. Consequently, the NoC load is 7.14%.

The benchmarks of each workload are run on the Gem5 [21] simulator separately. An ARMv7 operating at 550MHz, with 32kB split L1 cashes and an external DDR3 memory are employed. A trace with memory accesses of each application is captured and fed into OMNeT++ [17], where delay from accesses to shared resources, such as the NoC and the DDR3, are appropriately introduced. The resulting topology and mapping are shown in Figure 3.2. The traffic at the highlighted router R_6 (the closest router to the memory which is fully connected) is then recorded in a trace for each workload scenario, which will be used for the power simulations. After the flit-level traces have been generated for each workload scenario, testbench files were created. The latter read flits traces coming at each input port of the router, divide each FLIT further into four Phits, and inject them into the router.

Table 3.1.: MiBench applications for *Data intensive* and *Mixed workload* scenarios

Data intensive (run time= 20ms) (avg. 0,00615 access/cycle) 1 access every 162 cycles		**Mixed workload** (run time= 100ms) (avg. 0,00199 access/cycle) 1 access every 502 cycles	
Application	**Program**	**Application**	**Program**
Tiffmedian	[consumer]	FFT	[telecomm]
Jpeg-dec	[consumer]	Patricia	[network]
Patricia	[network]	Susan	[automotive]
Tiff2bwn	[network]	GSM	[telecomm]
Basicmath	[automotive]	SHA	[security]
Tiff2rgba	[automotive]	Basicmath	[automotive]

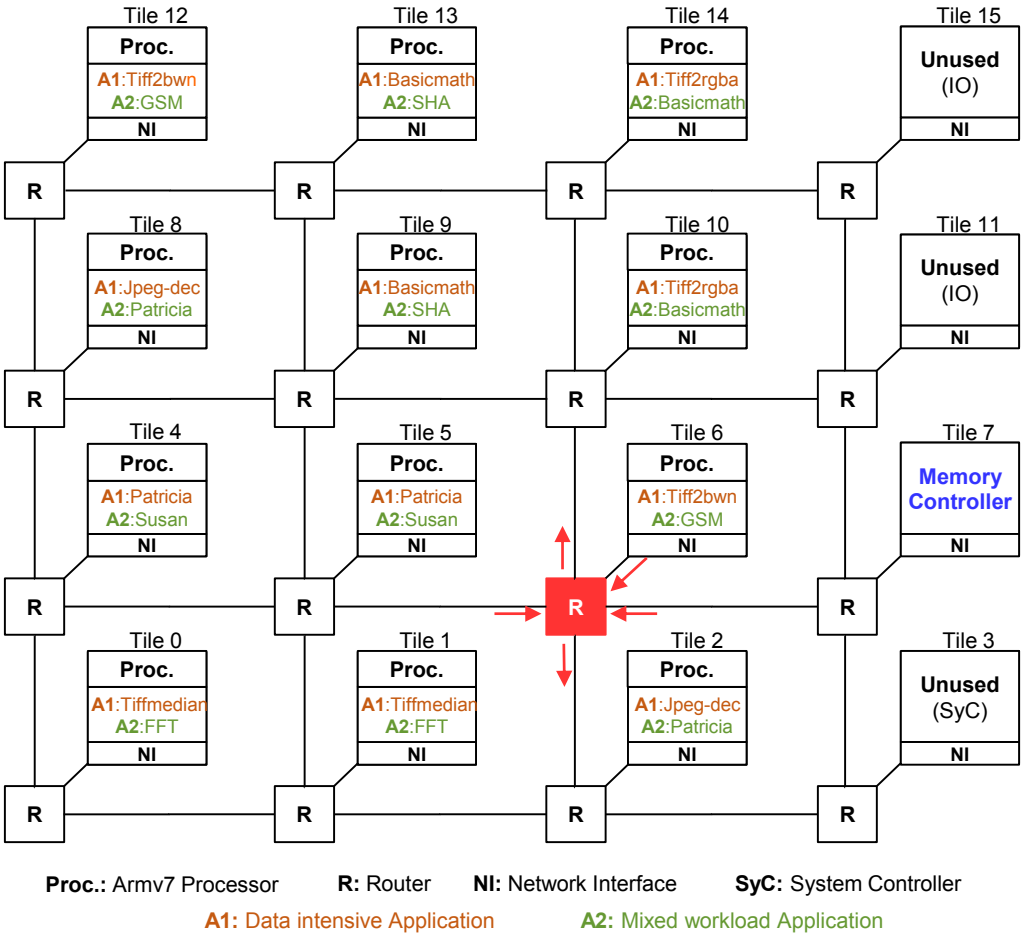


Fig. 3.2.: MiBench applications mapping on 4×4 NoC

Switching Activity and Power Calculation

Upon the generated traffics, Gate-Level simulation of the synthesized router was carried by Mentor Questasim 10.4c simulator to generate Switching Activity Interchange Format (SAIF) files. These files contain the switching activity information, which determines the toggle (switching from 0→1 and from 1→0) rate of all nodes, signals, nets, input and output ports of the circuit. Upon the generated SAIF files, the power was calculated with Synopsys PrimeTime (as depicted in Figure 3.1). Here, the accurate averaged power with bit-level accuracy is calculated for each component of the router.

This basic framework can easily be generalized from router level to evaluate the power of the complete NoC in different sizes, employing a variety of desirable applications.

3.2.2. Experimental Evaluation

This section provides an accurate power figures of the NoC routers designed for ASIC, where a realistic, more accurate, power estimation is provided by employing the introduced power analysis framework (cf. Section 3.2.1). In addition, as NoC must handle noise and errors that might occur in its infrastructure, we also illustrate the power overhead of a resilient real-time router (soft errors handling) and compare it with a baseline router (no error handling in the network) in terms of area and power. After that, power behavior under errors is investigated , where the power consumption of error recovery and packet retransmission mechanisms is evaluated to give an overview of the power behaviors under normal and abnormal NoC operations.

The errors and corruption may occur anywhere in the network, making fault-tolerant NoCs an important topic in safety-critical and high availability applications. The authors in [128, 22] give a good overview of existing work in fault tolerant NoCs, addressing both permanent and transient errors, also called hard and soft errors respectively. We focus on soft errors since they are more likely to occur. Almost all research work has focused on general purpose systems and upon error occurrence, the error recovery is realized by retransmitting lost or corrupt packets. Fault-tolerance approaches based on such error recovery mechanisms cannot be applied in real-time designs, which require that all errors and their effects must be known and properly handled [73]. However, retransmission protocols do not work if soft errors cause static effects and, therefore, lead to the failure of the network. Hence, we focus on fault-tolerance mechanisms that can be used in real-time systems and can handle static effects caused by soft errors. The power overhead of using reliable routing algorithms and additional control logic to harden a NoC was investigated in [126, 93]. However, they only use random traffic to estimate the power. While this is sufficient for a general insight on the power overhead, random traffic can lead to inaccurate and pessimistic results compared to a real system behavior. Moreover, online fault detection and location for NoC interconnects is introduced in [65]. They propose a resilient router, which employs Code-Disjoint Detection (CDD) scheme as an error recovery and they compare it with Switch-to-Switch (S2S) and End-to-End (E2E) schemes, introduced in [117].

A resilient real-time NoC architecture has been proposed and evaluated for FPGA in [130]. The architecture is resilient and predictable. That is, soft errors present only transient effects (resilient) and those effects are limited in time and in scope (predictable). We employ the resilient router architecture of [130] to develop our design for ASIC and evaluate area and power. Figure 3.3 presents

an overview of the resilient router, where changes to the baseline architecture (cf. Figure 2.3a) are highlighted. Resilience is achieved with three mechanisms: Fault Containment (FC) and Resilient Router (RR) at router level, and Reliable Transport (RT) at Network Interface level. FC and RR responsible for recovering from error in time, thus, ensuring that resilience and predictability are satisfied. RT responsible for guaranteeing the data integrity and delivery. FC is represented by Ingress Filters and CRC Generators (red blocks), which are added to the input ports of the router. The filter is responsible for deciding whether the FLIT is valid and may be safely propagated or not. If not, the error-affected FLIT is dropped before altering the router's state. The filter depends on Cyclic Redundancy Check (CRC) codes and Last Output Port (LOP) data to detect the error. LOP checking helps the router to know whether the FLIT comes from the correct output port of the upstream router or not. The CRC generator block generates the new CRC code for Head-Flits and Single-Flits due to route rotation.

To recover from errors that might occur outside or inside the router, the RR mechanism has been introduced [130]. It hardens the router's components to prevent static effects from transient faults and to allow the router to continue responding independently from the success of the previous arbitration cycle. In the figure, we highlight the components with major changes (VCAC and the Control Buffer), implementing the new resilient VC flow control. Based on the design of the above-presented resilient real-time NoC for FPGAs, we derive our resilient real-time router design for ASICs.

Power Results

Let us first present the total quiescent power (i.e., no traffic) consumed by the router. Figure 3.4 details the leakage and the dynamic power for baseline (3.7mW in total) and resilient (4.4mW in total) routers. As anticipated, the extensions for resilience lead to a higher power consumption. This idle period power is mainly caused by the activity on the clock pins of high-level non-clock gated synchronous elements.

Next, we probed the dynamic power under application-driven trace-based workloads using two different scenarios: *Data intensive* and *Mixed workload* (cf. 3.2.1). Figure 3.5 illustrates the switching activity and the clock tree powers per router, where we see now the total power increases with the higher load. Figure 3.5 also shows that, under *Data intensive* scenario, which provides higher loads, the resilient router consumes 15.63% more power than baseline. Overall, the power overhead is quite reasonable under 65nm technology library and higher load scenarios. Moreover, in order to show the inaccuracy between the estimated

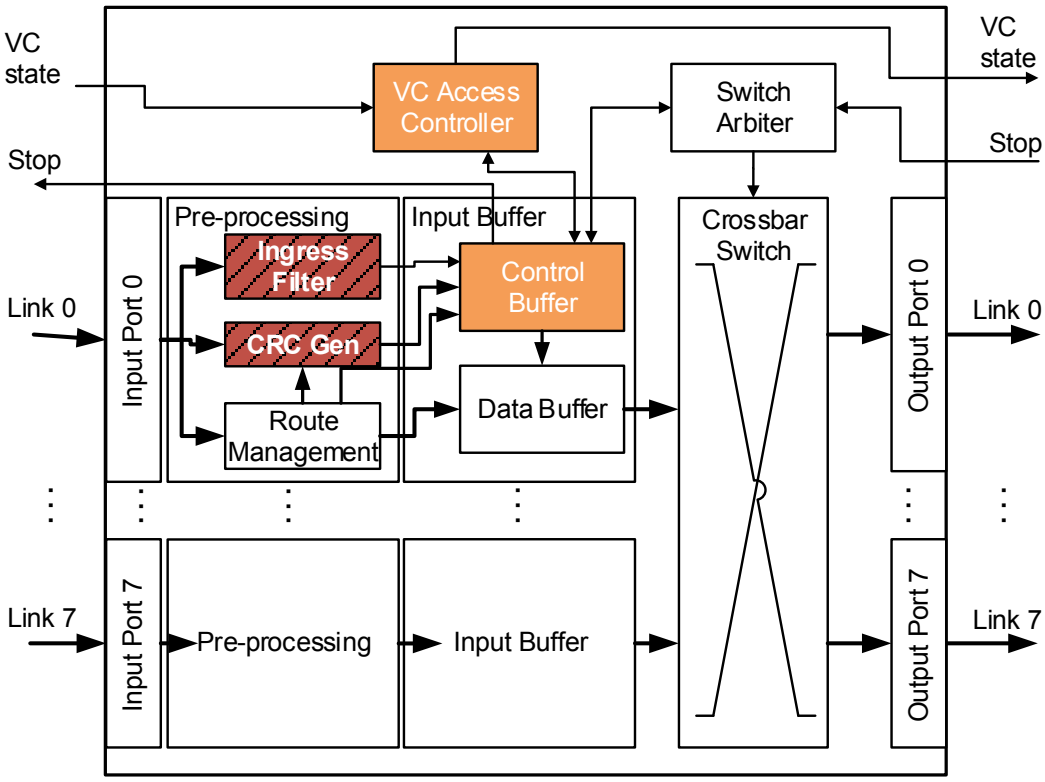


Fig. 3.3.: The resilient router architecture

power (done automatically by the tool) and the load-dependent power, Figure 3.5 also plots the estimated power per router. One can notice that the inaccuracy (the difference between the estimated and realistic power) increases when the router is more complex (resilient case). Also, the estimated switching power is equal or less than the *Data intensive* switching power, where the PrimeTime applies a default toggle rate and static probability on the all nets of the circuit which does not reflect the reality.

Another important aspect is to evaluate the area overhead of both baseline and resilient routers in ASIC. Figure 3.6 details the area and registers overhead for each component in both routers. The total resilient router size increases by 14.3% of the baseline router, and the total number of additional registers is 5.14%. These results indicate that the overhead of hardening the NoC using FC and RR mechanisms introduces tolerable cost w.r.t. area. On the other hand, the FPGA implementation of the resilient router indicates that the overhead to achieve resilience is 93.7% additional LUTs, 70% additional registers (considering data buffers as BRAMs), and 37.68%more power under full load with random uniform traffic. From this, we see an inefficient resilience overhead when we optimize the NoC for FPGA that might lead the designers to develop a real NoC component inside FPGAs.

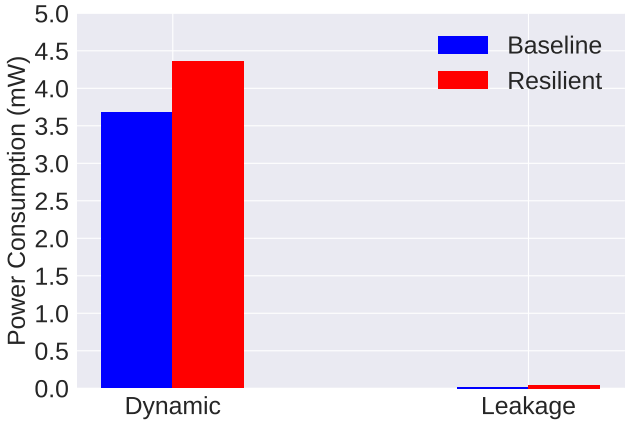


Fig. 3.4.: Routers'quiescent power

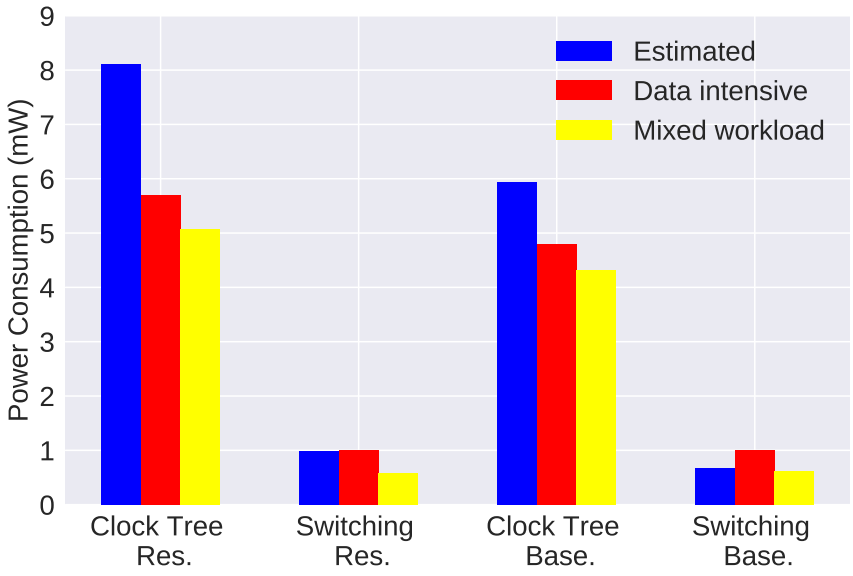


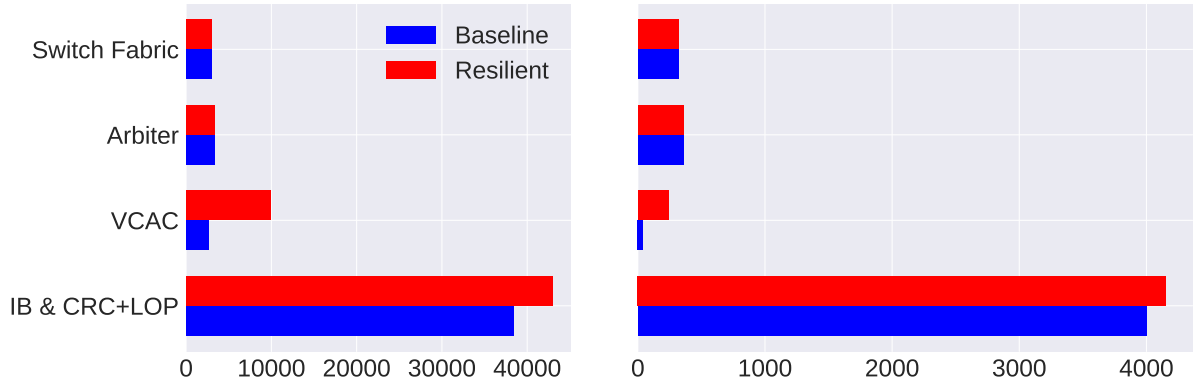
Fig. 3.5.: Estimated and load-dependent power of Resilient (Res.) and Baseline (Base.) Routers

Power Analysis under Errors

Although resilience is a highly desirable feature in a design, it must not incur vast costs, especially in terms of power. Hence, upon the introduced power analysis framework (cf. Section 3.2.1), we evaluate the power overhead under errors and break it down into error detection, error recovery and packet retransmission.

Error-Detection Power Overhead

The error detection overhead corresponds to the additional power consumed by the ingress filter (FC mechanism components: CRC, LOP and dropping-packet logic) and by the resilient components (RR mechanism: hardening VCAC and CB) in error-free case. The error detection overhead is the price paid during the NoC operation, regardless of error occurrences. This overhead is already



(a) Area (square micron) (b) Registers (#)

Fig. 3.6.: Baseline and Resilient routers area overhead employing ASIC design flow

reported in Section 3.2.2, where it has been analysed under data intensive and mixed workload scenarios.

Error-Recovery Power Overhead

The error-recovery overhead corresponds to the additional power consumed by the router in the error-case to recover from this error and prevent it from propagating. According to the Failure Mode and Effects Analysis (FMEA) and upon the functional error model derived in [130], which addressed the impacts of soft error at each component of the router separately, we employ randomly and intended bit-flip error injections inside and outside the router in order to track the power behavior under errors. Next, we evaluate the power consumed by the Fault Containment and Resilient Router mechanisms separately.

The Fault Containment mechanism (FC): Error-affected routing data could lead to HF, BF or TF corruption. Figure 3.7 presents the power consumption under different flits corruptions. For reference, the power in error-free case is also plotted.

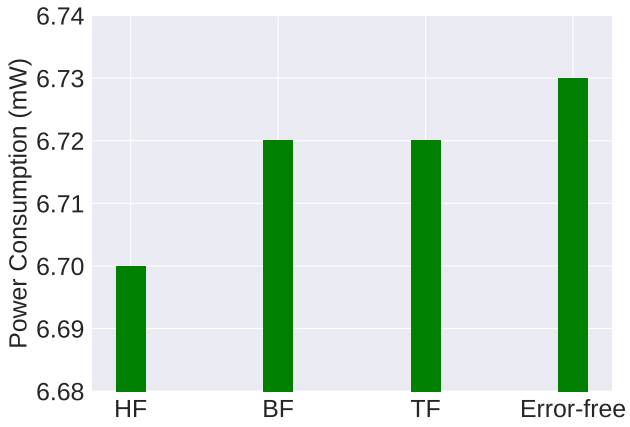


Fig. 3.7.: Power consumption under different corrupt flits

With Fault Containment, the router under errors consumes less power than in the error-free case. That is due to the dropping-FLIT logic, which confines the corrupt FLIT and drop it preventing the faulty FLIT to affect the internal state of the router. Figure 3.7 also shows that the HF corruption introduces the lowest power overhead since the whole packet is dropped. Additionally, the power overhead of handling a detoured FLIT has the same impact of FLIT corruption because the derouted FLIT will also be dropped by the LOP logic.

The Resilient Router mechanism (RR): Table 3.2 illustrates the power overhead of error recovery for each error-affected component separately. An error affecting the VCAC would lead to either improper VC release or reservation, which is in turn generally interpreted as Head-less or Tail-less packet respectively. An error affecting the Input Buffer (IB), particularly the Control Buffer (CB), which is responsible of memory access management, would lead to either improper read or write, leading to a FLIT loss; or to an inconsistent buffer state, which results in a reset of CB state. The Switch Fabric (crossbar) recovery mechanism is zero, because its state is reset at each arbitration cycle. The Arbiter recovery mechanism has negligible power overhead and is not shown. Overall, the router power under errors either remains the same or decreases (negative overhead). The former because the error recovery does not trigger any additional logic, as the router automatically recovers from errors at every cycle. The latter because corrupt packets are dropped and are not further processed by the router, resulting in less traffic internally.

Table 3.2.: Recovery power overhead under RR mechanism

Affected Block	Error Impact	Recovery Power Overhead (%)
IB $\rightarrow$ CB	Improper r/w, Inconsistent buffer state	0 0
VCAC	Improper release (Head-less) Improper reservation (Tail-less)	- 0.45 - 0.15
Arbiter	lost or detoured FLIT	0 (Automatic Recovery)
Crossbar	corrupt, lost and detoured FLIT	0 (Automatic Recovery)

Packet-Retransmission Power Overhead: The most important factor of power

overhead is the power to provide packet delivery guarantees and ensure the proper functionality under errors. To ensure packet delivery, the Reliable Transport (RT) mechanism at transport layer is employed, implementing the Stop-and-Wait (SNW) protocol [136]. Here, the sender NI waits for an acknowledgement signal from the receiver NI before sending the next packet, or retransmitting it after a timeout. Moreover, to compute the worst-case power overhead in case of error, we consider the negative acknowledgement signal as an additional signal, sent from the receiver NI to request the retransmission of missing packets. We derived our results at router level, where in error-case, the retransmission overhead is restricted to retransmit the negative acknowledgement signal, which is a single FLIT packet, and the missing or corrupt packets.

Figure 3.8 plots the power consumption of the three above-mentioned mechanisms at the router level in both *Data intensive* and *Mixed workload* scenarios, where the effects of different FLIT error rates on the power consumption are investigated. We define the FLIT error rate as the probability of a FLIT having a single bit-flip, resulting in higher error rates than expected in practice [9]. For simplicity, we plot the error recovery power consumption for both FC and RR mechanisms in only one line as, in the worst-case (tail-less packet), both of them incur the same recovery power consumption (cf. Table 3.2 and Figure 3.7). Also, we see the entire power consumption (FC+RR+RT) under high error rates induces acceptable power overhead compared with error-free case.

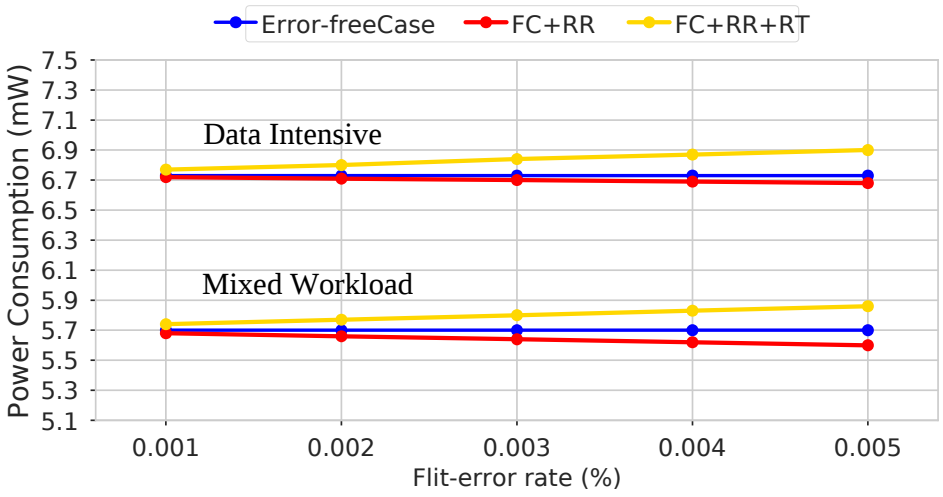


Fig. 3.8.: Power consumption under Fault Containment (FC), Resilient Router (RR) and Reliable Transport (RT) mechanisms

We have demonstrated the power overhead of real-time NoC routers designed for ASICs to get accurate power figures. In addition, the extra power overhead required to harden the router against errors is also investigated to give an overview

of the power behavior under normal and abnormal NoC operations. Overall, the NoC router presents high power consumption even under no-load operations (quiescent power in Figure 3.4) that requires an application of energy management and optimization schemes. In the following, we emphasise the NoC energy efficient design employing PANCs featuring multiple energy-savings schemes, which are especially needed in safety-critical hard real-time NoCs that are restricted to limited power budgets.

3.3. Energy Management Schemes

PANC may target individual/joint application of various energy management schemes, including *Power-Gating, Clock-Gating,* and *Dynamic Voltage and Frequency Scaling*, based on the energy-savings granularity required by the embedded applications. The joint application of the schemes features combined leakage and dynamic energy optimization.

Figure 3.9 depicts a high-level view of NoC augmented with energy-savings modules for *PG*, *CG*, and *DVFS*, whose functionality is controlled by PANC to optimize NoC energy. Routers are augmented with power switches (transistors), connecting routers to the power supply. These switches are controlled by PANC to turn routers on/off. In addition, routers support *CG* logical blocks, which gate/ungate the clock source based on PANC indications. Once PANC receives a NoC Req/Rel message from a sender, it goes through three steps to serve the message. First, it arbitrates between the requests to serve the highest priority one. Second, it processes the request by making a decision of the most efficient energy modes of the routers. Eventually, it translates these decisions into commands to the respective actuators in the NoC system, and acknowledges the sender. Noteworthy, PANC sends *PCG_OFF* (power/clock-gating off) messages to wake up all powered off routers (required by the sender) and activate their clock before it acknowledges a sender. Moreover, PANC signals the clock generator with the required clock mode derived from the current NoC traffic. The clock generator then provides NoC with the respective clock frequency, leading to a frequency increase/decrease *(DVFS)*. Besides, once PANC detects an idleness state of certain routers, it signals them with *PCG_ON* (power/clock-gating on) messages to gate the respective power and clock [85].

In the following, we detail the description of PANC functionality targeting *PG*, *CG*, *DVFS*, or their joint application on NoCs.

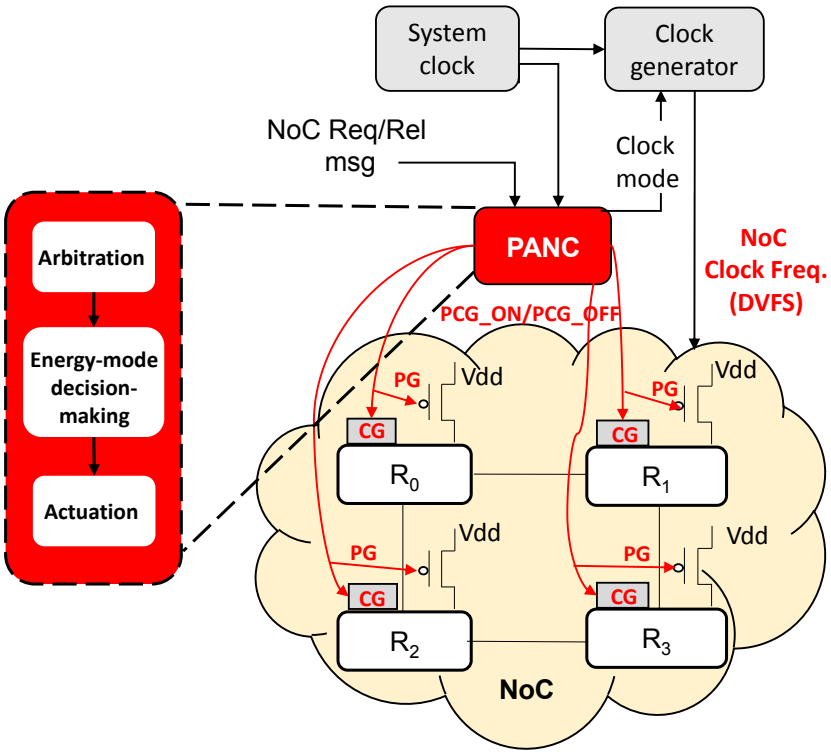


Fig. 3.9.: High-level view of NoC supporting modules of various energy-savings schemes, *PG*, *CG*, and *DVFS*

3.3.1. Power-Gating

This section customizes PANC to *Power-Gating* scheme featuring only leakage energy optimization [91]. The *Power-Gating* itself comes with additional timing parameters, the router wake-up latency (T_{WU}) and the *Break-Even Time (BET)*. T_{WU} corresponds to the additional latency the packet experiences along its path at each turned off router [91, 58], and *BET* rule corresponds to the additional cycles the router has to stay turned off in order to overcome the *PG* energy overhead. Both parameters are considered as major challenges in conventional *PG* methods. The wake-up latency can lead to a blocking of packets in the network at each router (i.e., the packet waits at each router until it is turned on), and thus to blocking propagation which can reduce the performance of the network. Also, this makes the traffic patterns more unpredictable, thus jeopardizing the temporal guarantees of critical tasks in hard real-time systems. One possibility to avoid this, is to turn on routers early, which then might decrease the energy-savings. Similarly, if a router is turned on again before the *BET* is over, the power overhead of turning a router on/off outweighs the power-savings and thus might even increase the overall power consumption. This leads to a trade-off between energy-saving capability and predictability (or performance) of the system.

Motivated by the aforementioned facts, we apply our global controller knowl-

Table 3.3.: Messages and Timing Parameters

Message Type	Message	Description
Power-Gating Messages	*PG_ON*	Power-Gating-ON: signal sent by PANC to turn-off a router by turning off the respective power switch
	PG_OFF	Power-Gating-OFF: signal sent by PANC to turn-on a router by turning on the respective power switch
Timing Parameters [77]	T_{WU}	Wake-Up (WU) latency: corresponds to the number of cycles required to charge up the local voltage of a powered-off router
	BET	Break-Even Time (*BET*): corresponds to the number of sleep cycles required to compensate the energy overhead induced by turning a router on

edge as an efficient factor to safely turn the routers on/off with accounting to these timing challenges. The approach covers different traffic patterns (e.g. periodic, sporadic). However, an optimization of PANCs in case of periodic systems is introduced, applying a grouping policy in order to better save energy, as introduced in Section 3.3.1. Table 3.3 presents an overview of the timing parameters and *PG* messages employed by PANC to configure NoC energy settings.

The global controller, PANC, gives a sender direct access when all routers along its path are *on*, or sends *PG_OFF* (Power-Gating-OFF) signals to wake up all powered off routers at a time. That way, it mitigates the cumulative waiting time the packet may experience along its path. Furthermore, PANC applies its knowledge of NoC global state to efficiently switch a router on/off. It filters out short router idle periods in order to overcome wake-up energy overhead. This can be easily fulfilled by considering the calculated *BET* [77, 119]. The conventional way only checks whether the router buffer and pipeline are empty, then sends a PG signal to turn it off. In our case, PANC uses a third factor. It checks whether there exists an active task that is going to use this router, and thus turning off the router might violate the *BET* constraint. If so, the controller keeps the router on.

In addition to that, PANC applies efficiently a router wake-up policy because it knows which router is powered-off, and sends a wake-up signal only to this router in contrast to sending it by the NI to the whole path [39]. The latter is not aware

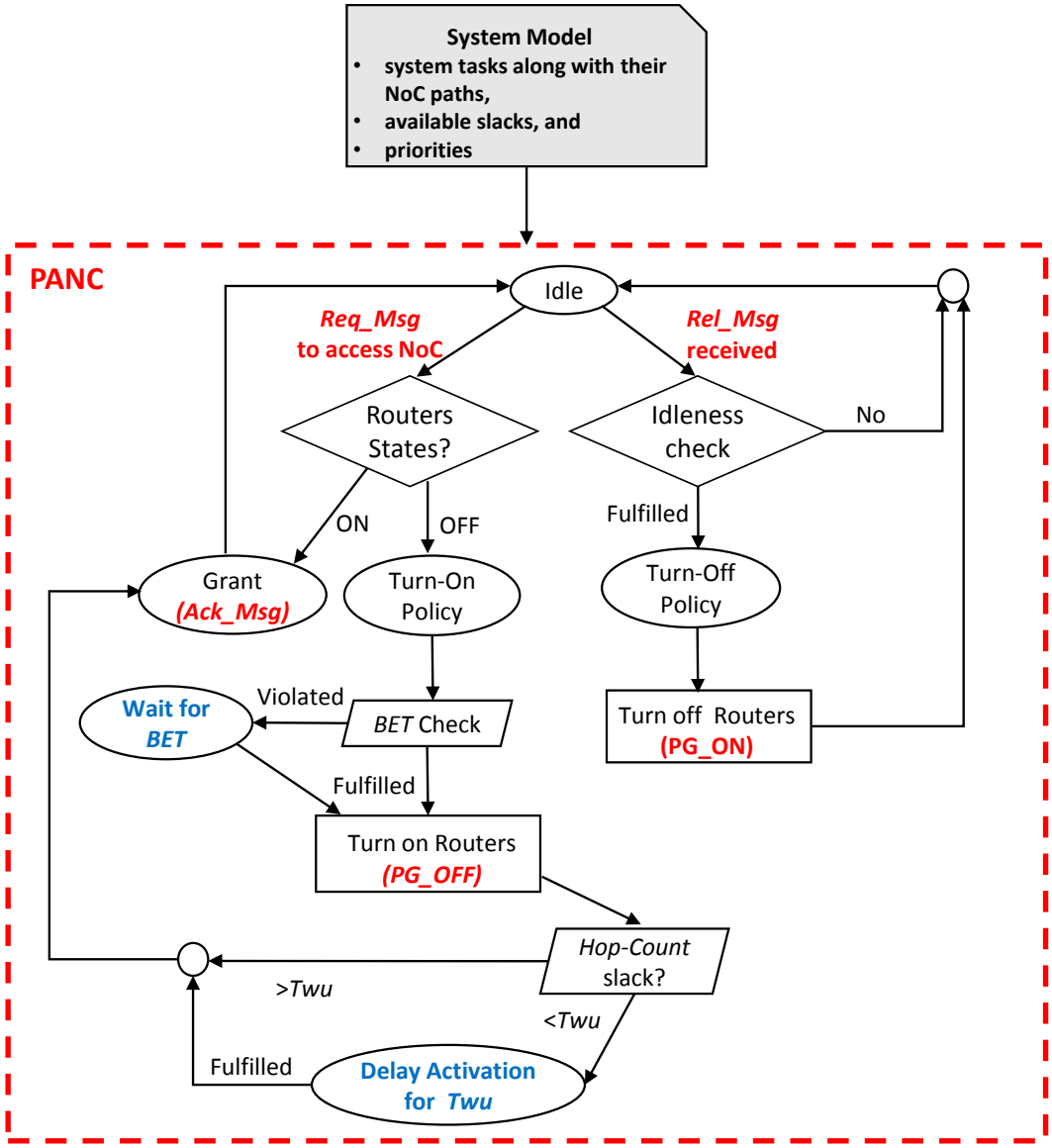


Fig. 3.10.: PANC workflow employing Power-Gating

of the routers' state along the packet path because its knowledge is limited to its sender activities. So when the NI has to send data, it sends a wake-up signal to the whole corresponding path to punch powered-off routers without first accounting for *BET*, and second for routers' state as they might be all *on* already. That, on the other hand and in addition to power penalty, requires monitoring multiple useless punch signals.

In the sequel, we detail the mechanisms of PANC used to save energy with safe latency overhead. Figure 3.10 summarizes the overall workflow of PANC where the messages in Table 3.3 are highlighted in red, and the timing parameters are highlighted in blue.

Low-Power Safe-Latency Turn On Policy

As previously described, PANC employs the AER model to turn off routers (or to keep them powered off) during the task's execution phase. Moreover, two other mechanisms are employed to overcome violating *BET* rule, with less and safe worst-case packet latency overhead: the *deadline slack* and the *Hop-Count slack*.

Deadline Slack

When PANC receives a request *Req_Msg* from an active task and figures out that at least one of the routers in the corresponding path is off, it checks for how long the router has been turned off. If it is not adequate (less than *BET*), PANC safely delays the task NoC access in order to consider *BET* rule. In real-time systems, each critical task is characterized by its relative deadline under which a task should be fulfilled. PANC exploits the task's deadline slack $DSlack_i$ in order to delay the task NoC access, assuring *BET* and thus saving higher energy. We remind that the $DSlack_i$ defines the time budget between the task's deadline and its worst-case response time accounting for PANC latency overhead, computed later in Section 4.1. Thus, the predominant condition to apply the delay policy to account for *BET* is to have positive slacks of the requesting tasks. Furthermore, when PANC is supposed to turn on a powered-off router that has been turned off for less than *BET*, it checks the available task's slack and delays the task to a number of cycles ($DelCyc_i$) based on the following equations:

$$Roff_r = BET - OffCyc_r \tag{3.1}$$

$$DelCyc_i = \begin{cases} Roff_r, & \text{if } DSlack_i > Roff_r \\ 0, & \text{otherwise,} \end{cases} \tag{3.2}$$

where $Roff_r$ denotes the number of off cycles required to keep the router powered off to account for BET; $OffCyc_r$ denotes the number of *off* cycles the router has been turned off. That is, PANC delays a task only if the available slack is greater than the required off cycles ($Roff_r$).

After checking *BET*, PANC inactivates power gating by sending (in parallel) power-gating-off signals (PG_OFF), in order to bring the power supply back to the corresponding routers. Simultaneously, it investigates sending a grant to the active task using the following *Hop-Count* slack mechanism.

Hop-Count Slack

It defines how far the powered-off routers are from the source tile in terms of hops. PANC exploits the *Hop-Count slack* of the powered-off routers in order to

mitigate the WU latency the packet may experience along its path. The number of hops that completely hide the *WU* latency of a certain powered-off router is given as follows:

$$N_{hops} = (\lceil \frac{T_{WU}}{T_R + T_L} \rceil), \tag{3.3}$$

where T_R is a router delay, and T_L is a link delay, considering that the packet takes $(T_R + T_L)$ cycles per hop (router and link) [91].
For example, if the powered-off router is the third hop along the packet's path, then a hop-count slack of 2 is typically able to compensate 10 cycles wake-up latency for routers with a 4-stage pipeline. Thus, PANC investigates the *Hop-Count* slack of the powered-off routers along the corresponding path, and if all of them have adequate Hop-Count slack (none of them is one of first two hops in 4-stage router, or first 3 hops in 3-stage router), the PANC acknowledges the sender (*Ack_Msg*), while simultaneously turning on the powered-off router(s). However, when the *Hop-Count* slack is exceeded by at least one router, PANC has to delay the sender accounting for T_{WU}. At runtime, applying the delay policy to account for T_{WU} is always allowed as an offline analysis assured that the tasks have enough slacks to account for the T_{WU} delay (detailed later in Section 4.1.2). However, if the offline analysis indicated negative slacks for a certain task, the application of PANC must be excluded from routers along the packet's path of the corresponding violated task. That is, the corresponding routers are kept always *on*, and the violated tasks are not required to synchronize their NoC accesses with PANC – utilizing a direct access.

As we design our energy-savings approach to meet hard real-time objectives, the increased latency of a critical task must not jeopardize timing guarantees. Therefore, we always assure a safe application of the aforementioned task delay (induced by *BET*) by checking, at run time, the available slack against the required delay. Moreover, in order to demonstrate the diverse sources of additional latency overheads, Section 4.1 details the breakdown of the PANC latency overhead the packet may experience once it is issued.

Turn Off Policy

As PANC knows the global state of the NoC, it turns off the NoC routers based on the active tasks and consequently based on the traffic in the NoC. PANC has to ensure the idleness of routers along the packet's path. Therefore, as the approach conforms with timing guarantees, when at least one task is active, PANC waits for a release message from the destination tile, which ensures none of the routers

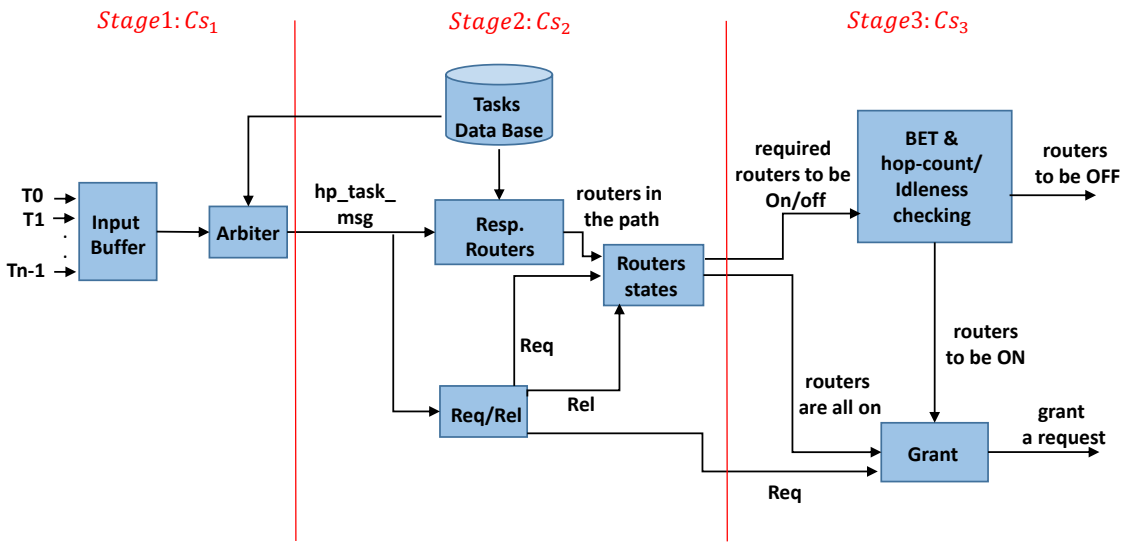


Fig. 3.11.: Block diagram of the PANC implementation

is still being used by the corresponding source (the last packet has been arrived). After that, it checks whether there exist other active tasks that are using the same routers. If so, it repeats the same strategy by waiting for their arrival messages, otherwise, it turns off the idle routers.

Block Diagram of PANC

Figure 3.11 presents a high level view of the PANC implementation and its processing pipeline for handling requests. Overall, PANC is pipelined in three stages. Arriving requests are stored in an input buffer. An arbiter, in the first stage, selects the highest priority request. In the next two stages, PANC decides the best power state of routers. That is in turn derived from the workflow of PANC (cf. Figure 3.10). Thus, the second stage corresponds to Req/Rel checking, deriving the respective routers in the task path, and revealing the required routers to be on/off; the third stage corresponds to *BET* and hop-count/idleness checking in case of Req/Rel, respectively; and then the grant process to acknowledge a request.

In the sequence, we optimize PANC for periodic pattern, exploiting the periodical feature, in order to better save power using the activations grouping policy.

PANC Optimization for Periodic Patterns

The control layer employing PANC covers different types of traffic patterns, e.g., *periodic* and *sporadic*. Moreover, it involves different cases of deadline occurrence, e.g., a task has its deadline equal to its period $D_i = T_i$ (implicit-deadline),

and the deadline is smaller than the period $D_i < T_i$ (constrained-deadline). Despite the fact that PANC can follow the same general policy to acknowledge a request, we exploit the periodicity in order to achieve more efficient energy-savings. When a purely periodic system is running, and thus PANC knows precisely the next arrival of a task activation, it follows a grouping policy to turn on/off routers and consequently acknowledges the request.

Grouping Tasks Activations

As the router switching from $off \longrightarrow on$ is considered one of the most important factors overall Power-Gating mechanisms because of its energy overhead (e.g. capacitance charge), we investigate how to decrease the number of the router switching activities employing the periodical feature. Figure 3.12 depicts a periodic task activations scenario, and the PANC reaction in normal case and under Optimized PANC (Opt-PANC). In normal case, the router has to be turned on at every new activation arrival, which in turn leads to a significant *PG* energy overhead. To this end, we investigate to efficiently group task activations to decrease the aforementioned overhead. Therefore, at design time, we seek to group the activations in the hyper-period interval after which the periodic pattern of all tasks is repeated [137]. It applies the resulting grouping model for all other activations after the hyper-period. The maximum number of shifting cycles of a task activation ($Shift_i$) can be bounded by Equation 3.4:

$$Shift_i = D_i - R_{\{i,PANC\}}, \tag{3.4}$$

where $R_{\{i,PANC\}}$ denotes the worst-case response time of a task i including the latency overhead of PANC in case routers are turned off.

The resulting grouping, as it is depicted in Figure 3.12, corresponds to shift one activation, while fulfilling directly the right next one, decreasing the number of the switching activities to the half. Next, PANC should be augmented with the tasks periods and their shifting parameters, resulting from the optimization at design time. At runtime, PANC reschedules the tasks arrivals based on the state of the former ones. That is, it checks the state of the former activation, in case it has been shifted, it fulfills the current activation once it occurs. In contrast, it shifts the current activation in case the former one has not been shifted.

Moreover, as the grouping policy designed for the worst-cast response time of a task activation, PANC has to intelligently decide whether the router has to stay on until the next activation occurs (cf. Figure 3.12, the red dashed box), or turn it off. Thus, at runtime, PANC always compares the time difference between the next activation arrival of any task, and the idleness start of the router, i.e., if all active tasks that utilize the router have arrived to their destinations. The comparison can

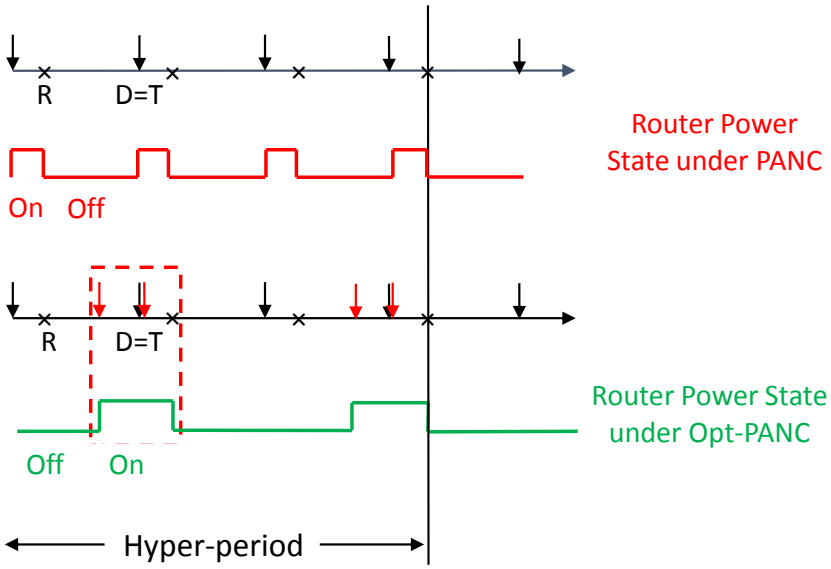


Fig. 3.12.: A periodic task activations scenario and the corresponding router power state under PANC (red color) and under optimization (green color)

be expressed by the following formula:

$$\forall i \in \{ROT\}, \;\; NxtAct_i - IdleStart_r > BET, \tag{3.5}$$

where $NxtAct_i$: denotes the next activation of any task i, which belongs to the set Router Overlap Tasks (ROT);
$IdleStart_r$: denotes the idleness start of the router r.

PANC employs the Equation 3.5 and turns the router off only if the *BET* rule is not violated (Equation 7 is fulfilled). In the latter case, the router's state in the red dashed box may adopt an additional transition (turned-off until the next arrival occurs to be on again) in order not to lose power.

The Opt-PANC can decrease the Power-Gating power overhead at high data rates with short message sizes as it is demonstrated later in the experimental evaluation (cf. Section 4.2.2). However, the packet latency will significantly increase from the fact that PANC shifts a task activation until the next one. We remind that, in hard real-time systems the utility of finishing a critical task too early is the same of completing it by its deadline [148]. Consequently, we only need a formal timing analysis of the system tasks to prove the feasibility of PANC, i.e., no deadline misses occur, as introduced later in Section 4.1. Note that for purely periodic systems the deadline typically corresponds to the period and hence an activation can be delayed until shortly before the next activation. In addition, in case the analysis provides temporal guarantees, the usage of Opt-PANC is then up to the user and his application requirements.

3.3.2. Dynamic Voltage and Frequency Scaling

In this section, PANC features Dynamic Voltage and Frequency Scaling scheme, targeting leakage and switching energy-savings [90, 98]. PANC optimizes *DVFS* for NoCs hosting workloads with hard real-time requirements. That is, it assures all hard deadlines are met while applying *DVFS*. Figure. 3.13 exemplifies the usage of dynamic frequency scaling comparing with the static one. It depicts three senders ($AppA - AppC$) with different execution lengths (indicated with a bars) which are jointly using the NoC . If a static energy management is employed, the energy mode (frequency and voltage) of the system is constant (indicated with a red, dashed line), i.e., the frequency must be high enough so that no deadline for transmissions from critical tasks is endangered. Therefore, whenever senders are not running in parallel, for instance due to the transient errors, arrival jitters, changes in communication volume, or conditional executions of senders, the static energy management results in significant energy losses.

Conversely, in our approach, whenever a critical sender is activated, or terminated, the Energy Mode (EM) of the system may change, i.e., frequency and voltage alterations can be done to safely adjust the energy required for chip operation. Consequently, in the example from Fig. 3.13 the maximum frequency (f_3) is necessary only for very short periods of time. Therefore, dynamic adjustments can significantly decrease average energy consumption.

Note that in the context of safety-critical real-time systems adherence to the timing requirements is the most important condition, i.e., it dominates the improvement in energy management. Therefore, the sender may start its transmission only if the frequency has been adjusted to the requirements of all other currently active senders, i.e., all running senders must finish their execution on time. It is necessary to assure that the newly introduced traffic (transient load) will not cause overload – delay packets which started their transmissions in different EM leading to missed deadlines. In addition, the frequency switching due to physical properties of a chip is not instantaneous. For accounting to the introduced issues mode change latency has been introduced – delaying switching on senders until the frequency settles, as detailed later in Section 4.1.

The number of simultaneously active tasks defines the maximum frequency required to transfer their transmissions. This allows the designer to calculate the safe EM settings for each individual set of active tasks at design time (offline). These settings are available for PANC during runtime, from which it safely adjusts the NoC energy. Calculating the EM settings is introduced in Section 3.3.2.

Now, we describe the mechanism conducted by PANC to safely apply *DVFS*. As previously described, PANC relies on the global state of NoC workload, pro-

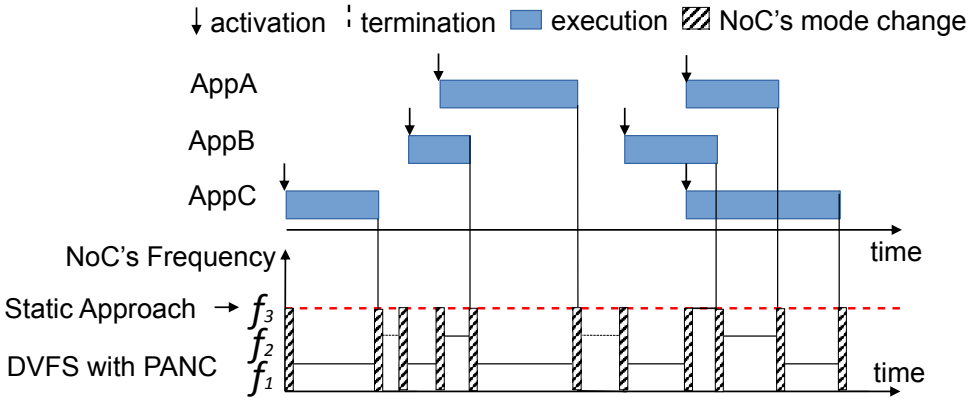


Fig. 3.13.: Comparative example of power consumption in the NoC with the static energy mode and *DVFS* with the control-layer for three synchronized applications.

vided by the protocol-based control messages between the on-core supervisors, the clients, and PANC. Each of the request message *Req_Msg* and release message *Rel_Msg* may initiate the transition of the system into a different EM – manifested with a change of frequency and the corresponding energy levels. The responsibility of PANC is to assure that transitions between modes are safe. This is done with a stop message *Stp_Msg* and a resume message *Rsm_Msg*. In the worst-case, each supervised task causes two mode-changes per activation (i.e., activation and termination).

Besides, in the worst-case, it may happen that the decrease of the frequency influences all other already running transmissions which were initiated in the previous scenario with a frequency adjusted to a different set of tasks. Therefore, PANC delays the mode change until all traffic initiated during the previous scenario safely leaves the NoC . Otherwise, newly injected packets may cause transient effects through delaying other packets running in the same mode, and vice versa. In order to prevent that, PANC first sends the *Stp_Msg* to all active senders. After receiving this message, senders must stop initiating new transmissions until they will be resumed once again by PANC. Next, PANC waits until all messages leave the NoC . Afterwards, it enforces the new NoC frequency via a clock generator (see Figure 3.9), establishing NoC new energy mode. Eventually, after the new energy settings are settled, PANC resumes the stopped tasks.

Obtaining Settings for EMs

The basic latency of each flow depends on the NoC frequency. This implies that the response time of flows also depends on the NoC frequency. The aim is to assure the minimal energy level which guarantees a safe running of concurrent active tasks. One approach is to create EM settings for each possible combination

of active tasks. Assuming that there are x tasks in the system, the number of possible EMs is $x!$, which can be prohibitively expensive in some practical cases. Therefore, the decision regarding the number of EMs is a trade-off between the analysis complexity and the EMs switching overhead on the one side, and the energy efficiency of the system on the other. In our case, we define one EM for multiple potential combinations of the system tasks, whose transmissions fall in the same frequency domain without missed deadlines.

Let us assume that $\mathscr{A}_s$ is a subset of all tasks $\mathscr{A}$. Consequently, the relevant flow-set becomes $\mathscr{F}_s$, which is a subset of $\mathscr{F}$, and which consists only of flows belonging to tasks from $\mathscr{A}_s$. Now, assuming a given frequency ψ_k, R_i would be the worst-case transmission time of a flow ϕ_i for this particular $\mathscr{F}_s$, as detailed later in Section 4.1. At this moment, the main challenge becomes how to find an optimal frequency for a given $\mathscr{F}_s$. Given the fact that saving energy is one of the main objectives in the real-time embedded domain, the optimal frequency becomes the minimal one which allows all flows to still meet their deadlines. Algorithm 1 demonstrates how to find it.

First, the flows $\mathscr{F}_s$ constituting one EM are provided as an input. Additionally, a granularity of the search is controlled with the input parameter $diff$. The smaller value implies a more frequency efficient but also more computationally complex search method. Then, the lower and the upper bounds are assigned their initial values (line 1 in Algorithm 1). First, the maximum frequency ψ_k is considered the current frequency (line 2 in Algorithm 1) and the schedulability is tested for all flows treated in a decreasing manner with respect to their priorities (lines 3-12 in Algorithm 1). If a schedulability cannot be reached even with the maximum frequency the system is rendered infeasible (lines 13-15 in Algorithm 1). Conversely, while the difference in the bounds exceeds the parameter $diff$, the search process is conducted. The algorithm starts and updates the values of the current frequency, lower and the upper bounds in a form of a binary search. Besides, when the taskset is schedulable, the analysed frequency ψ_k is updated with the lower found one. Finally, when the distance between the bounds becomes less than $diff$, the analysed frequency is assigned to the ψ_{opt} and that value is returned (lines 30-31 in Algorithm 1).

Algorithm 1: Finding Optimal Frequency($\mathscr{F}_s$, $diff$)

input : $\mathscr{F}_s$ – set of flows belonging to the analysed $\mathscr{A}_s$
input : $diff$ – granularity for bound search
output: ψ_{opt} – optimal frequency for the analysed $\mathscr{A}_s$

1 $u\_bound \leftarrow \infty$; $l\_bound \leftarrow 0$;
2 $curr \leftarrow \psi_k$;
3 $sort(\mathscr{F}_s, P, \downarrow)$;
4 $unschedulable \leftarrow false$;
5 **foreach** $(\phi_i \in \mathscr{F}_s)$ **do**
6 $\quad \mathscr{F}_i^s \leftarrow find\_infs(\phi_i)$
7 $\quad R_i \leftarrow cmpR_i(\phi_i, \mathscr{F}_i^s, curr)$
8 $\quad$ **if** $(R_i > D_i)$ **then**
9 $\quad\quad unschedulable \leftarrow true$;
10 $\quad\quad break$;
11 $\quad$ **end**
12 **end**
13 **if** $(unschedulable)$ **then**
14 $\quad$ **return** $infeasible$;
15 **end**
16 **while** $(u\_bound - l\_bound > diff)$ **do**
17 $\quad curr \leftarrow \frac{u\_bound - l\_bound}{2}$;
18 $\quad sort(\mathscr{F}_s, P, \downarrow)$;
19 $\quad unschedulable \leftarrow false$;
20 $\quad$ *check schedulability with curr frequency*
$\quad$ // (*repeat instructions from lines* $5 \rightarrow 12$)
21 $\quad$ **if** $(unschedulable)$ **then**
22 $\quad\quad l\_bound \leftarrow curr$;
23 $\quad$ **else**
24 $\quad\quad$ **if** $curr < \psi_k$ **then**
25 $\quad\quad\quad \psi_k \leftarrow curr$;
26 $\quad\quad$ **end**
27 $\quad\quad u\_bound \leftarrow curr$;
28 $\quad$ **end**
29 **end**
30 $\psi_{opt} \leftarrow \psi_k$;
31 **return** ψ_{opt};

3.3.3. Integrated Energy Control

This section illustrates the higher efficiency of PANC featuring two combinations of energy-savings schemes: Power-Gating and Clock-Gating (*PG+CG*), and (*PG+CG+DVFS*), targeting both leakage and dynamic energy optimization [90, 85, 87]. Figure 3.14 depicts the general workflow of PANC, which basically consists of processing of two basic messages from the client: *request* and *release* messages. We remind that PANC also contains a database consisting the system model that comprises the required inputs, e.g., tasks properties and the respective frequency for every set of tasks, whose computation is presented in Section 3.3.2.

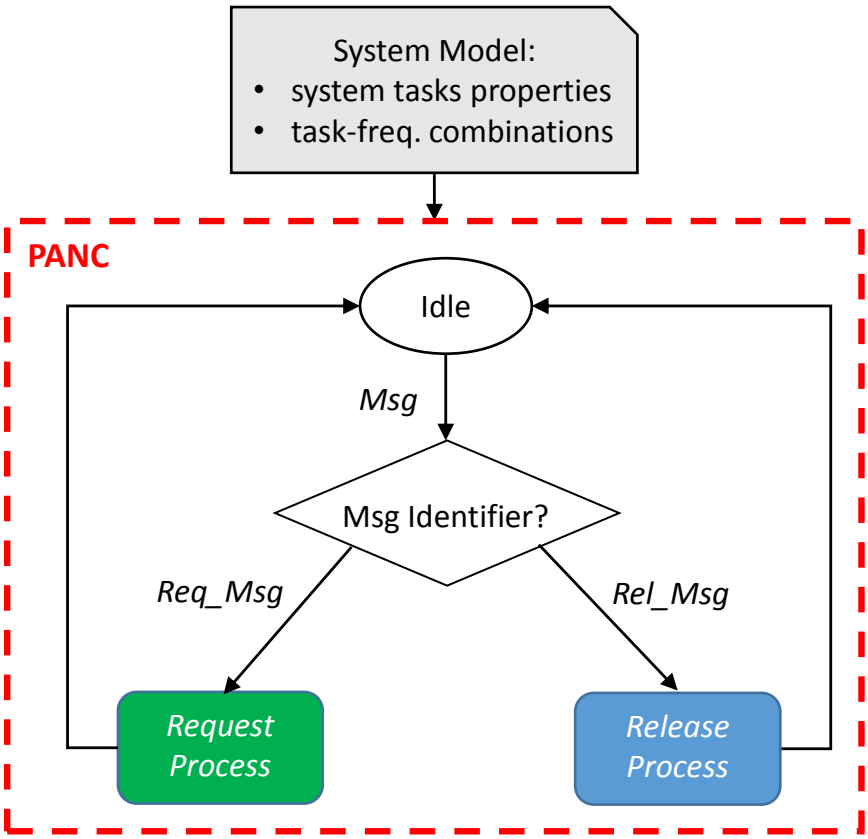


Fig. 3.14.: PANC general workflow

PG+CG Scheme

In this scheme, PANC features both *PG* and *CG*, investigating leakage and clock-tree energy-savings [90, 85]. It basically relies on the time slots under which it turns on/off routers to simultaneously activate/gate the clock of the routers. As previously mentioned, all routers are augmented with a *CG* block that gates/activates the clock based on the PANC indications. Thus, for example, by sending

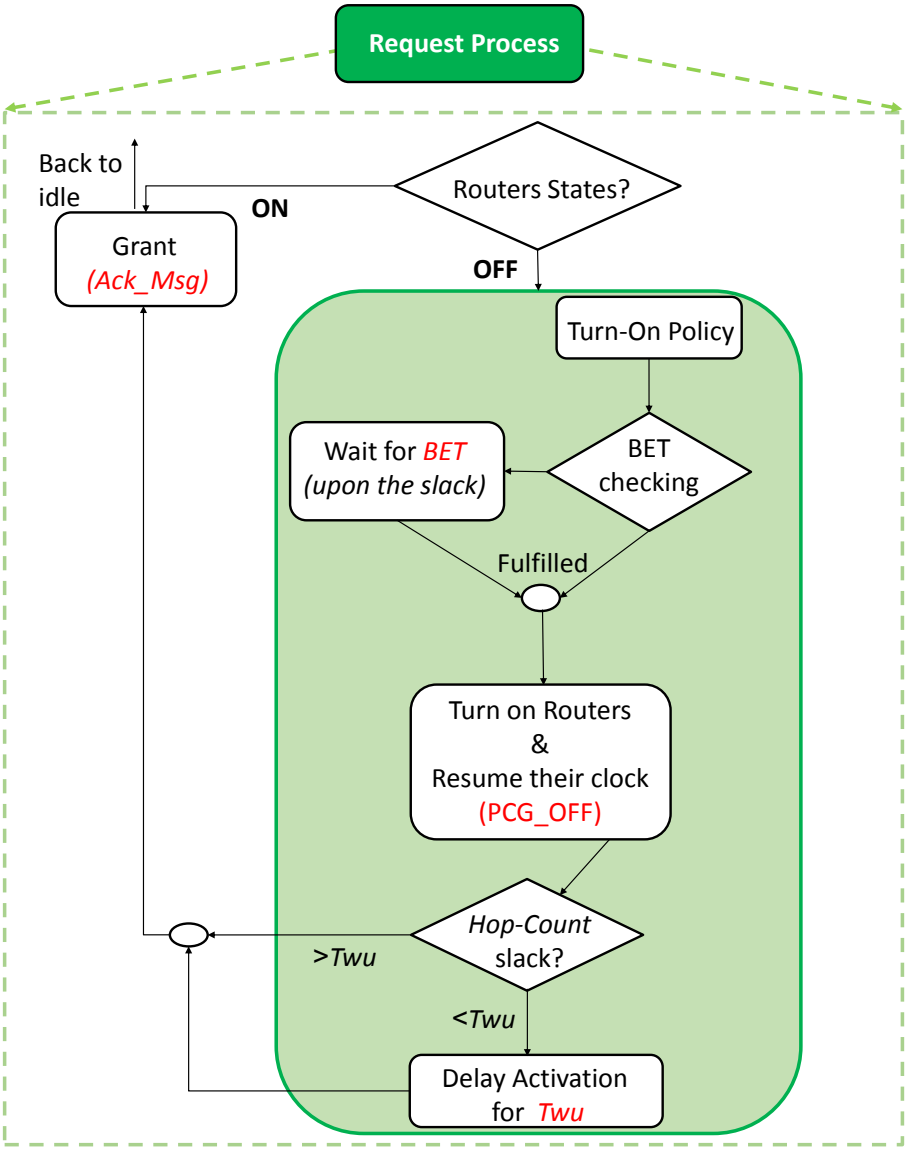


Fig. 3.15.: PANC sub-workflow of processing a sender request message employing PG+CG scheme

a power/clock-gating on *PCG_ON* message to a certain router to apply *PG+CG* scheme, PANC turns off the router by cutting down the power supply (leakage savings), and gates the clock by the *CG* block as long as the router is powered off (clock-tree savings). As PANC knows the global state of the NoC based on the synchronization protocol with clients, it effectively monitors the power states of NoC routers, and thus applies power/clock gating upon tasks' permissible slacks. Figure 3.15 depicts the workflow of PANC to process a request message employing *PG+CG*. When PANC receives a *Req_Msg* from a sender, it first checks the routers states along the sender's routing path. Routers could be all *on* (indicated by *ON* in Figure 3.15), or at least one/all of them are powered off (*OFF* in Figure 3.15). PANC directly acknowledges the sender to access NoC

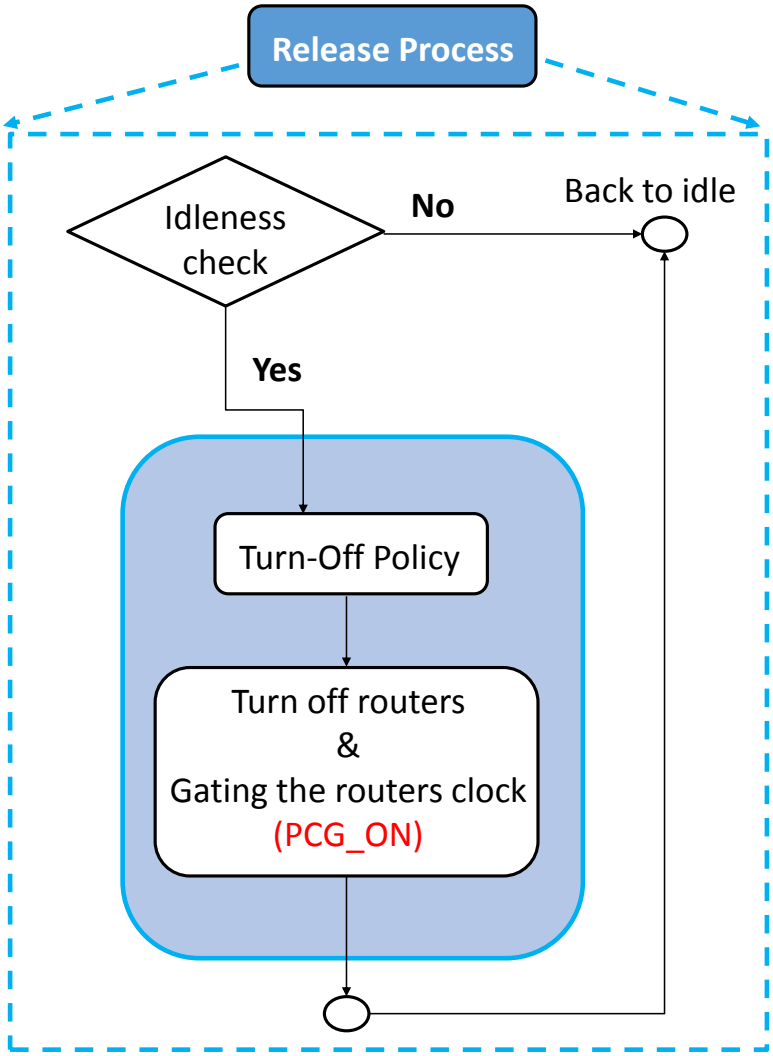


Fig. 3.16.: PANC sub-workflow of processing a sender release message employing PG+CG scheme

when all respective routers are *on*, or it safely employs a turn-on policy. The latter corresponds to turn on routers after accounting for *BET* factor, where PANC may keep the router off for additional cycles to account for *BET*, based on the available *task's deadline slack* $DSlack_i$, as previously detailed in Section 3.3.1. Afterwards, PANC inactivates *PG*/*CG* by sending (in parallel) power/clock-gating-off signals (PCG_OFF) to turn on the corresponding routers and bring their clock back. At the same time, PANC investigates sending an Ack to the requested task based on the *Hop-Count* slack, exploited by PANC to mitigate the wake up latency (T_{WU}) the packet may experience along its path (detailed in Section 3.3.1). Thus, based on the exploited hop-count slack of the powered off/clock-gated routers, PANC acknowledges the sender (Ack_Msg).

Furthermore, upon receiving a a release message *Rel_Msg*, PANC must ensure the idleness of routers along the packet's routing path, as depicted in Figure 3.16.

In case of *(idleness=No)*, PANC goes back to idle – waiting for release messages from other tasks that still use the corresponding routers. Otherwise, *(idleness=Yes)*, it applies the turn-off policy by gating the power and the clock sources than the idle routers, turning them off.

PG+CG+DVFS Scheme

In this case, PANC features *PG*, *CG*, and *DVFS*, targeting both leakage and dynamic (clock-tree and switching) energy-savings [90, 85]. Here, Figures 3.17 and 3.18 imply the required extensions to additionally adopt the *DVFS* functionality. Upon receiving a *Req_Msg*, PANC checks the router states and whether the frequency increase is required with increasing the number of active tasks. Figure 3.15 shows various potential outputs of the checking stage. In case of routers are *on* and frequency scaling is required *(ON & Freq. Scaling)*, PANC applies the *DVFS* scheme, highlighted in green, in order to safely reconfigure NoC , and finally acknowledge the client request to access NoC (Grant process). Recall, to apply *DVFS*, PANC first conducts *Stp_Msgs* to stop all current active tasks and waits for all initiated packets to leave the NoC , avoiding transient effects of overload. Afterwards, PANC enforces the new NoC frequency (derived from its database) and eventually resumes the stopped tasks conducting *Rsm_Msgs*, as detailed in Section 3.3.2.

In case of *OFF and frequency scaling*, PANC exploits the waiting time, required for initiated packets to leave NoC , in order to fulfill turn on policy. Note that PANC, during the waiting time, only fulfills the *BET* checking without turning routers on. After enforcing the new energy settings via updating the frequency, and resuming the active senders, PANC turns the respective routers on, reactivates their clock, and acknowledges the request after checking the *hop-count* slack. That way, it is guaranteed that routers are turned on only after the new frequency is adjusted.

In case of release message, PANC checks the idleness of the respective routers and whether the frequency decrease is required under decreasing the number of active tasks (cf. Figure 3.18). The checking stage results in four cases. In case of routers are idle and no scaling is required *(Yes & No Freq. Scaling),* PANC turns routers off, gates the respective clock, and goes back to idle. However, when frequency scaling is required, PANC additionally employs *DVFS* before it goes to idle. In case of routers are not idle and scaling is required (No & Freq. Scaling), PANC applies *DVFS* and goes to idle.

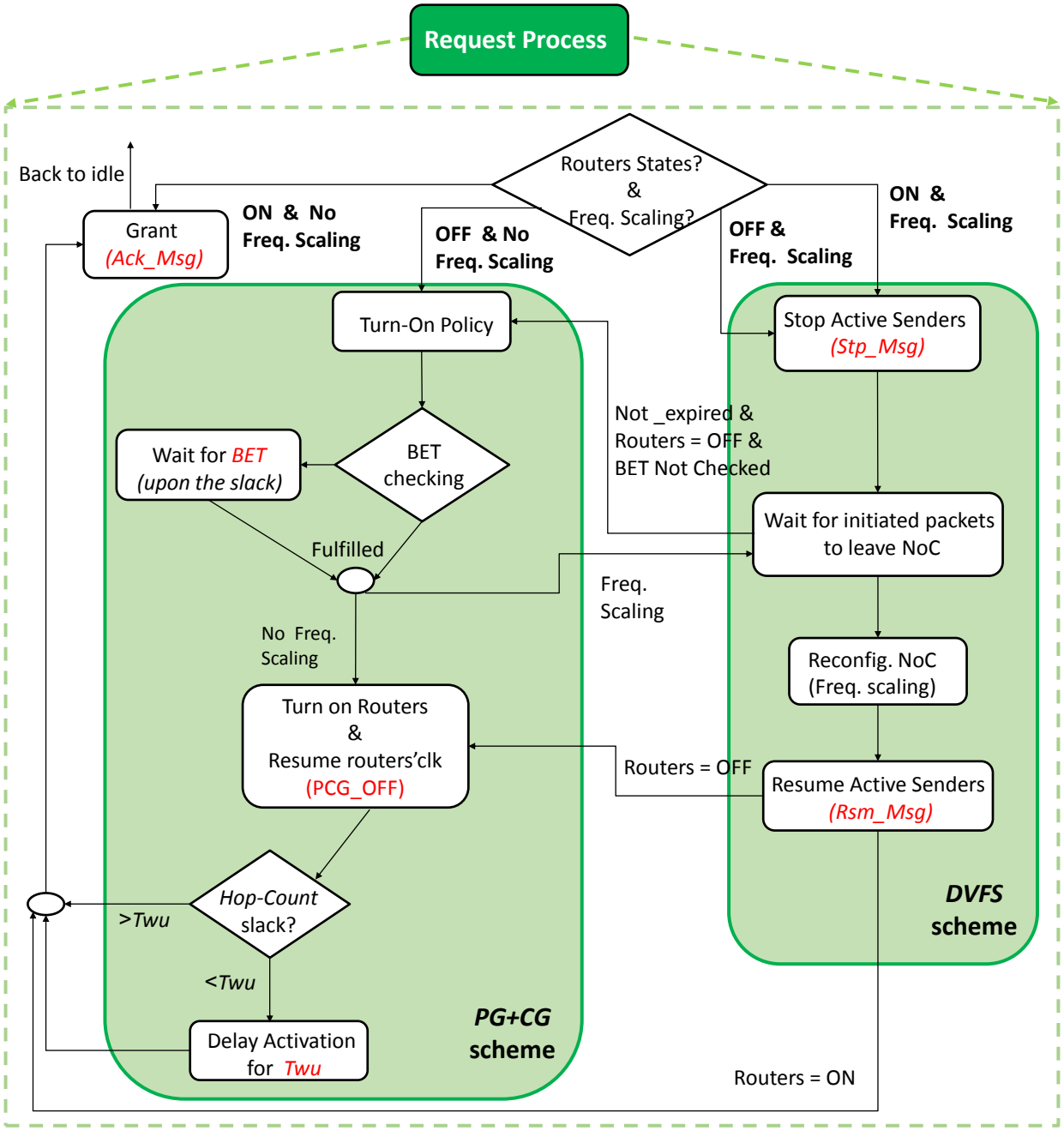


Fig. 3.17.: PANC sub-workflow of processing a sender request message employing PG+CG+DVFS scheme

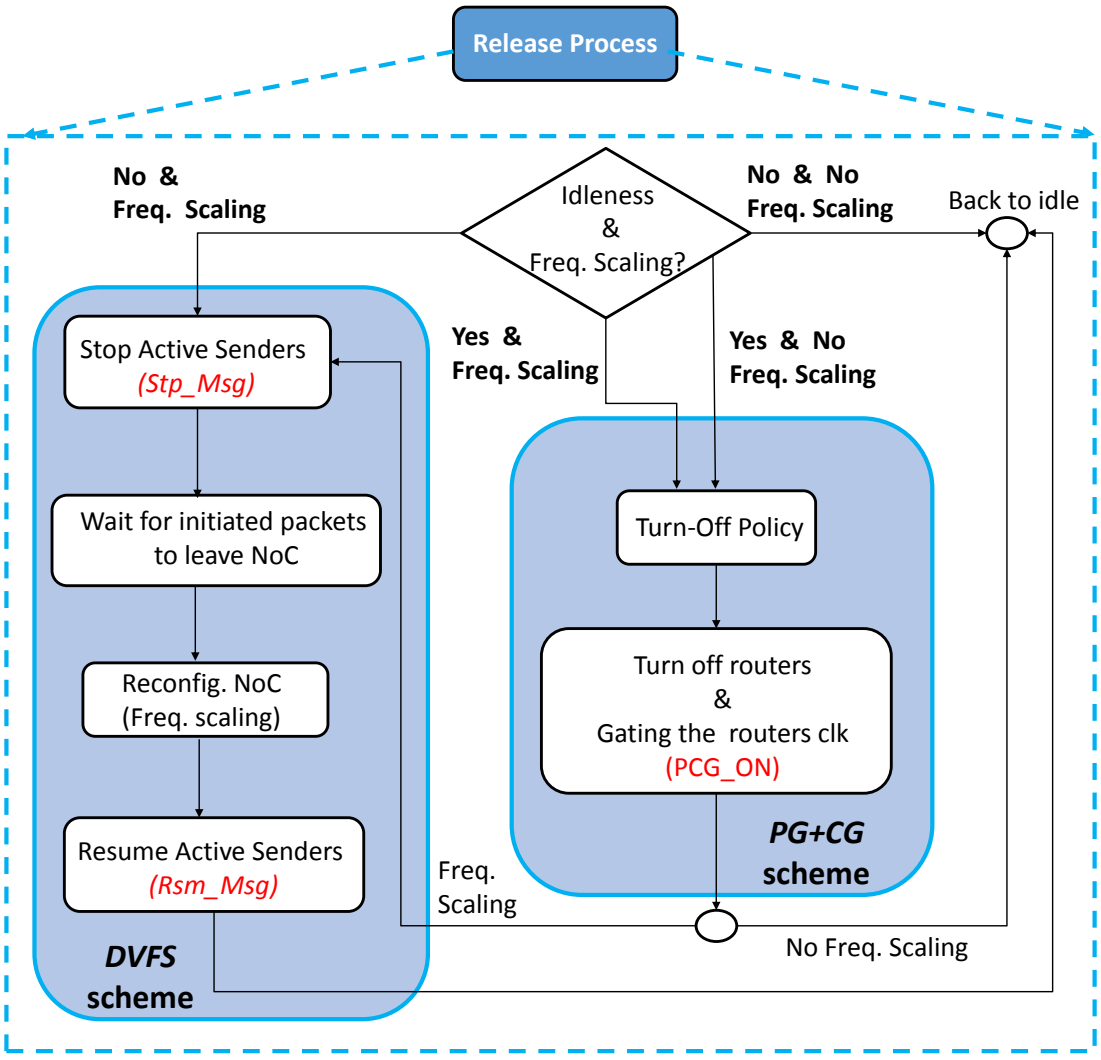


Fig. 3.18.: PANC sub-workflow of processing a sender release message employing PG+CG+DVFS schemes

3.4. Summary

Networks-on-Chip designed for hosting real-time applications require both efficient real-time guarantees and tight energy limitations. The efficient and safe energy management is crucial for the success of MPSoCs in the market of safety critical and real-time systems. This chapter first evaluated the power overhead of real-time NoC routers designed for ASIC to investigate accurate power figures. Next, it introduced an approach for dynamic energy management of hard real-time NoCs to manage the energy loss. The approach corresponds to the control layer which saves energy of NoC routers accounting for changing system behavior. In addition, the control layer complies with the real-time requirements, while adjusting NoC energy settings. The control layer employing energy-savings

schemes is decoupled from the traffic-control conducted interdependently in the baseline network routers. This design choice has been adopted to consider the presented requirements for developing the control layer in safety-critical hard real-time systems (cf. Section 2.2.1). That is, it makes the approach transparent to the running applications and the baseline arbitration conducted in the NoC data layer, allowing bounded (low) interference and assured real-time constraints. Besides, the isolation between NoC data and control layers is highly considered by some commercial NoC products like the Kalray MPPA [25, 46] and the Tilera Tile64 [151, 16].

The control layer is basically composed of central units, PANCs, and on-core local supervisors, the clients. It applies a protocol-based synchronization to support PANC with the current system state via the clients. PANCs, in turn, have the capabilities to feature individual/joint application of different energy-savings schemes to optimize various sources of energy dissipation. In this work, Power-Gating, Clock-Gating, and Dynamic Voltage and Frequency Scaling have been employed, targeting both leakage and dynamic (clock-tree and switching) energy-savings. Note that PANC is a user-friend approach. That is, setting PANC to feature individual energy-savings scheme or integrated schemes like *(PG+CG)*/*(PG+CG+DVFs)* is highly flexible and dependent on the energy-savings granularity desired by the user.

PANC's functionality relies on the introduced Acquisition-Execution-Restitution (AER) task model, where NoC energy optimization can be performed during tasks' execution phases. In addition, two existing slacks Hop-Count slack and task's deadline slack have been exploited to improve the application of energy-savings schemes in hard real-time NoCs. The permissible slacks of critical functions are extracted (at design time) and exploited by the PANCs (at runtime) to monitor the NoC states, and thus applying the potential energy-savings schemes – providing application temporal guarantees. PANCs, by applying their knowledge of the system state and by exploiting the aforementioned slacks, turn effectively routers on/off after accounting for *BET* rule, resulting in high efficient *PG* approach and advancing the related work. PANCs have also been optimized for periodic traffic, fulfilling higher energy-savings. Besides, Clock-Gating has been efficiently applied through gating the clock source during the time slots on which the routers are considered off. Next to this, we proposed a novel mechanism, which allows to apply *DVFS* for the majority of existing NoCs in safety-critical domains without complex hardware extensions. The improvement comes from the dynamic voltage and frequency adjustments performed at runtime, depending on the global system state.

Furthermore, as we target hard real-time many-core systems which are quite

ubiquitous in, e.g., automotive domains, the scalability of the control layer through PANCs has been demonstrated by employing multiple PANCs in large NoCs. Consequently, an additional control NoC has been developed to communicate between local PANCs without influencing the conducted data transfer in the underlying data layer. Thus, energy efficient NoC design is achieved, while fulfilling the presented objectives and satisfying temporal requirements for integration of the control layer with hard real-time systems (Section 2.2.1).

Chapter 4: Formal Verifications and Energy-Savings Evaluations

For evaluation of NoC based MPSoCs running specific realistic usecases/benchmark suite, there are basically two available tools: simulation and analysis. The simulation has been used by the state-of-the-art, e.g., BookSim [84] and Noxim [33]. However, it is often hard to get stimuli covering all potential corner-cases, i.e., the actual worst-case scenario. To remedy this, besides simulation, we employ formal, model-based verifications to prove the compliance of the introduced control layer to hard real-time requirements. The process is basically based on formal temporal analysis frameworks used for the underlying NoC architecture, e.g. [168, 108, 155]. The employed analysis framework [79, 123] is then extended to evaluate the temporal overhead of the control layer and verify it against the tasks deadlines.

Figure 4.1 illustrates the formal verifications and the evaluation process of the NoC energy management employing the control layer as an extended version of Figure 2.6. As previously described, the temporal analysis engines are responsible to provide information about the schedulability of hard real-time tasks at design time, accounting for the control layer (cf. Section 4.1). After that, PANC employs the provided timing properties to trade off between energy-savings and real-time requirements using the OMNeT++ network simulator [17] (cf. Section 4.2). During runtime, PANC generates traces that reflect the observed NoC active times. That is, the trace contains varying time slots under which the NoC router is reported as on/off with the respective frequency. The NoC energy-savings extraction block is based on the observed traces and the baseline NoC energy consumption in order to derive and report the NoC energy-savings figures. The baseline NoC energy is derived using ASIC design flow employing Synopses tool chain for both 65*nm* UMC and 28*nm* Taiwan Semiconductor Manufacturing Company (TSMC) process technologies [91, 90, 85].

The energy-management of NoC using the control layer featuring the individual/joint application of the energy-savings schemes (*PG*, *CG*, and *DVFS*) is

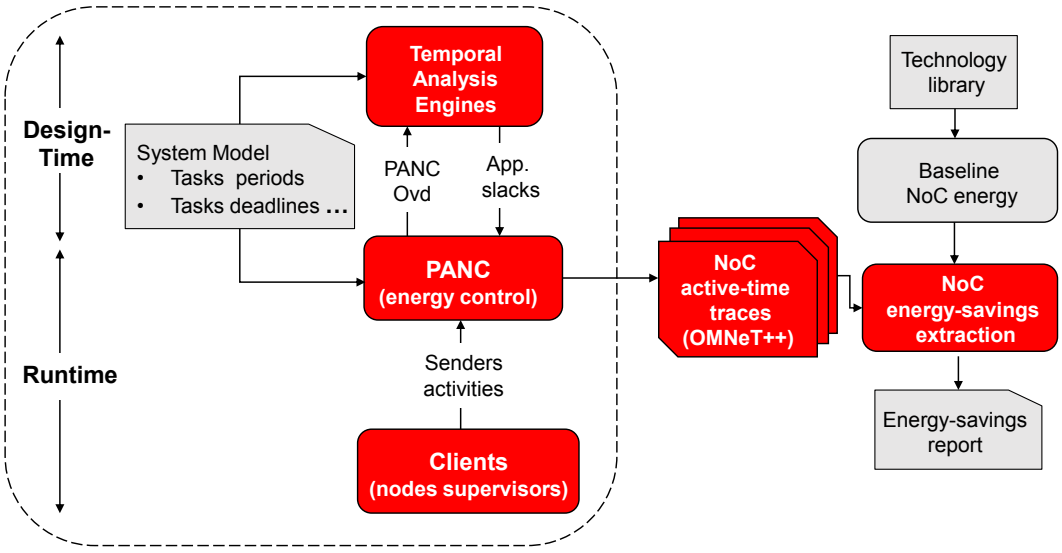


Fig. 4.1.: The comprehensive framework for NoC energy management evaluation

evaluated employing simulation and formal analysis tools to respectively evaluate the average and the worst-case scenarios. Besides, various usecases are utilized, (i) the control of a vehicle with assistance functions usecase from automotive domains [90, 85, 98], (ii) the Full Authority Digital Engine Control (FADEC) usecase from avionic domains [91], (iii) various benchmarks and synthetic workloads [91]. The experimental results indicate that PANC achieves significant energy-savings compared with a baseline NoC (none of the energy-savings schemes has been applied), as well as outperforms the conventional related work schemes as presented in Section 4.2.

4.1. Temporal Analysis

This section demonstrates the applicability of the control layer for hard real-time systems, where all tasks are characterized with their firm deadlines. This is tackled through the following steps:

- We employ a NoC analysis framework, introduced later in Section 4.1.1, to compute the worst-case response time of a transmission via NoC .
- The framework is then extended to develop a mathematical model and describe the temporal behavior of the control layer (cf. Section 4.1.2). That provides a formal analysis framework to compute upper-bounds on the response time for transmissions, accounting for the control layer, making the approach predictable while efficiently saving energy.

- Eventually, we derive the permissible deadline slack (i.e., the time budget between the transmission response time and its deadline, see Definition 1), where the safe applicability of PANCs is verified with positive slacks, satisfying temporal guarantees requirement (*Req1* from Section 2.2.1) [91, 98, 90, 85].

Here, we introduce the used baseline system and analysis model. We assume a multiprocessor platform Θ using a 2D mesh NoC interconnect with $n \times n$ routers. At each router there is a tile housing resources (e.g., processing cores). The time a router needs to route a head-flit is denoted as d_R with $d_R = \frac{cyc_R}{\psi_R}$, where cyc_R denotes the number of cycles it takes a router to route one head-flit and ψ_R is the router clock frequency. Similarly, d_L with $d_L = \frac{cyc_L}{\psi_R}$ denotes the time needed to transfer one flit between two neighbouring routers, where cyc_L is the number of clock cycles for this transfer. The NoC clock frequency can operate on multiple frequencies between $\psi_{R,min}$ and $\psi_{R,max}$ enforced by PANCs as previously described in Section 3.3.2.

The workload of the platform is given by a set of tasks $\mathscr{A} = \{\alpha_1, \alpha_2, \ldots\}$, where multiple tasks can run on the same core. During an inter-arrival period, each task α_i performs potentially discontinuous on-core execution and network transmissions. Consequently, the NoC load of a core i can be modelled with a superposition of the traffic generated by the tasks on the core. In the following we distinguish between the properties of transmission (flows) and of the tasks on cores. We denote the latter with star as superscript. For example, we denote the latency of transmission as C where the on core execution time is denoted as C^*. The tasks/cores are divided into safety-critical and best-effort senders. SC senders are characterized with their respective deadlines D^*, which denotes the time limit to finish the execution and transmission. From this, a deadline D for the transmission can be derived. If a task α_i meets its deadline D_i^*, it is considered schedulable. If all SC tasks of $\mathscr{A}$ are schedulable, $\mathscr{A}$ itself is considered schedulable.

Network transmissions, hereafter also referred to as a flow ϕ_i, are a source of a sequence of packets. Each flow then is characterized by:

i. its source router ρ_i^{src},

ii. its destination router ρ_i^{dst},

iii. a path between these routers π_i, expressed as a collection of traversed links, which size is called the number of hops h_i,

iv. its size σ_i in the number of flits,

v. its minimum inter-arrival time T_i between two successive flows (i.e., two activations),

vi. its deadline D_i with $D_i \leq T_i$,

vii. its priority P_i, and

viii. its release jitter J_i^R.

A new transmission starts only after the period T_i of the previous transmission expired. Hence, D_i describes a conservative upper bound on the transmission latency. This simplified model ensures that if a flow α_i of each task is received before its deadline D_i, the entire task set $\mathscr{A}$ is schedulable.

4.1.1. Timing Analysis Framework

Despite the fact that the control layer is compatible with diverse NoC analysis frameworks like real-time calculus [168], network calculus-based frameworks [108, 26], integer linear programming based analysis [31], and compositional performance analysis [155], we employ the analysis from [79, 123]. It is an exemplary example that is easy to present. In [79, 123] the authors consider non-shared virtual channel between flows that interfere at any router output port; unique priority per flow; Static-Priority Preemptive (SPP) arbitration policy inside routers to resolve the congestion between tasks; and limited buffer sizes [79]. The number of virtual channels is related to the employed usecase. In our experiments, employing different usecases, we need 4 virtual channels (cf. Section 4.2.1).

Thus, we first introduce the basic worst-case analysis concepts used in our approach. For this, we define the traversal time of a packet of a flow ϕ_i without congestions, known as the *basic network latency* C_i [79, 123].

Definition 6. *The basic network latency C_i of a task* i *denotes the transmission time of an activated instance of a task* i *via the basic NoC in case of isolation (no congestion exists). It can be bounded as follows:*

$$C_i = \underbrace{(h_i - 1) \cdot d_R}_{\textit{header routing}} + \underbrace{h_i \cdot d_L}_{\textit{header traversal}} + \underbrace{(\sigma_i - 1) \cdot d_L}_{\textit{payload traversal}} . \tag{4.1}$$

The traversal time is equal to the time it takes a header flit to reach the destination router, followed by the traversal of the remaining flits across the last link (due to the pipelined transmission).

Definition 7. *The worst-case response time R_i of an activated instance of a task* i *in case of congestion is the sum of its basic network latency and the interference*

it suffers. It can be computed by iteratively solving the following equation:

$$\mathbf{R_i^{n+1}} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\mathbf{R_i^n} + J_j^R + J_j^I}{T_j} \right\rceil \cdot (C_j + B_j), \tag{4.2}$$

where $\left\lceil \frac{R_i^n + J_j^R + J_j^I}{T_j} \right\rceil$ denotes the maximum number of activations can be released by tasks that have higher priority than i ($hp(i)$) during the time interval $[0, R_i]$ and B_j accounts for the interference due to backpressure. The Equation 4.2 also accounts for release jitter J_j^R and indirect interference jitter J_j^I.

We find the variable R_i appears on both sides of the Equation 4.2. A solution can be computed iteratively [8, 79, 123], because all components grow monotonically with respect to the response time. The iteration starts with $R_i^0 = C_i$ and terminates when two successive iterations provide identical results $R_i^{n+1} = R_i^n$. The iteration also terminates if $R_i^{n+1} > D_i$, which means a deadline miss for this task. As the higher priority tasks may cause interference to the lower ones in case they share with each other the same/part of the network path, we sort the tasks based on their priorities and proceed to analyse them from the highest priority one by one. Finally, the schedulability test consists of checking whether the condition ($R_i < D_i$) holds for every task in the system.

To compute R_i, we need to define the sets of flows, which can cause interference to the flow under analysis. Let $\mathscr{F}$ be a set of all possible flows in the system and let $\mathscr{F}_i$ denote a subset of flows from $\mathscr{F}$, which can cause direct interference to ϕ_i. That is, $\mathscr{F}_i$ denotes all flows which have a higher priority than ϕ_i, and share (a part of) the network path with it. Formally:

$$\forall \phi_j \in \mathscr{F} \mid P_j > P_i \wedge \pi_j \cap \pi_i \neq \emptyset \Rightarrow \phi_j \in \mathscr{F}_i. \tag{4.3}$$

Next to the direct interference, a flow ϕ_k can cause indirectly interference to ϕ_i. That is, ϕ_k can delay a packet of ϕ_j and hence cause two consecutive packets of ϕ_j to appear and pre-empt ϕ_i in a time interval smaller than T_j. This effect can be modelled as an additional jitter J_j^I for stream ϕ_j [79, 123]. It can be derived by:

$$J_j^I = \begin{cases} R_j - C_j & \text{if } \exists \phi_k \in \mathscr{F} \mid \phi_k \in \mathscr{F}_j \wedge \phi_k \notin \mathscr{F}_i \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

Moreover, we also need to account for the backpressure effects B_j that influence the computation of R_i.

Definition 8. *Backpressure is a phenomena occurs when the router buffer space is limited. That is, when a router buffer overflows due to congestion, the router*

holds off the preceding (transmitting) router from sending any data packets until the overflow is eliminated [155, 165].

If ϕ_j inflicts only downstream indirect interference on ϕ_i, the backpressure has two upper bounds. One is the total amount of interference that ϕ_j can suffer downstream, and the other one is the traversal time across a single link of all flits of ϕ_j that can be simultaneously buffered within the contention domain (CD) of ϕ_i and ϕ_j [79, 123]. It can be computed as follows:

$$B_j = \sum_{\substack{\phi_k \in \mathscr{F}_j \\ downstream(\phi_i)}} \left\{ \left\lceil \frac{R_j + J_k^R + J_k^I}{T_k} \right\rceil \cdot \min\left\{ C_k + B_k, \beta \cdot |CD_{i,j}| \cdot d_l \right\} \right\}, \tag{4.5}$$

where β is the size of each Virtual Channel (VC) in the router buffer, and $CD_{i,j}$ represents the contention domain of ϕ_i and ϕ_j, i.e., the set of links shared by ϕ_i and ϕ_j. $Downstream(\phi_i)$ corresponds to the set of flows ϕ_k the interfere with the set of flows ϕ_j after the contention domain of ϕ_i and ϕ_j (i.e., after ϕ_j interferes with ϕ_i). Similarly, upstream indirect interference $upstream(\phi_i)$ corresponds to the set of flows ϕ_m that interfere with the set of flows ϕ_j before the contention domain of ϕ_i and ϕ_j [79]. If ϕ_j inflicts both upstream and downstream indirect interference on ϕ_i, the backpressure has a single upper-bound equal to the amount of interference that ϕ_j can suffer downstream. It can be computed as follows:

$$B_j = \sum_{\substack{\phi_k \in \mathscr{F}_j \\ downstream(\phi_i)}} \left\lceil \frac{R_j + J_k^R + J_k^I}{T_k} \right\rceil \cdot (C_k + B_k). \tag{4.6}$$

For more details on the worst-case analysis, a reader can consult the work of Indrusiak et al. [79, 123].

4.1.2. Control Layer Modeling with the Timing Analysis Framework

This section extends the introduced timing analysis framework to integrate the control layer latency overhead and derive the worst-case transmission time of real-time tasks, allowing safe applicability of energy-savings when all deadlines

are met. The latency parameters of the control layer are mainly induced by the control messages and the applied energy-saving scheme by PANC. The control layer topology employing different NoC sizes has been presented in Sections 2.2.2 and 2.2.5. We detail in the following the developed analysis framework to feature the employed energy-savings schemes: Power-Gating, Dynamic Voltage and Frequency Scaling, and integrated energy control [91, 98, 90, 85].

PG Latency Analysis

The Power-Gating itself comes with an additional latency overhead the packet may experience along its path [91]. We detail in the following the extension of the worst-case response time of a task on NoC when PANC features Power-Gating.

Definition 9. *The worst-case response time of an activated instance of a task i, accounting for PANC employing PG* $(R_i^{PANC(PG)})$ *can be bounded by the following formula:*

$$R_i^{PANC(PG)} = R_i + MLO_{PANC}^{Del}. \tag{4.7}$$

As described before, R_i denotes the worst-case response time of a task i using the baseline NoC (cf. Equation 4.2); MLO_{PANC}^{Del} denotes the Maximum Latency Overhead induced by PANC and the delay policy.

Definition 10. *The maximum latency overhead* MLO_{PANC}^{Del} *from the Equation 4.7, induced by the control layer through PANC, a task i may experience once it is issued can be bounded by:*

$$MLO_{PANC}^{Del} = R_i^{Ctrl} + R_i^{Proc} + R_i^{Del}, \tag{4.8}$$

where R_i^{Ctrl} denotes the worst-case latency overhead of the control messages sent to and from PANC; R_i^{Proc} denotes the worst-case processing time the task instance may experience by PANC; and R_i^{Del} denotes the worst-case delay the task instance may experience by PANC in case of turned off routers.

The control messages that cause latency overhead, as derived directly from the description of PANC employing *PG* (Section 3.3.1), are *Req_Msg* and *Ack_Msg* as follows:

$$R_i^{Ctrl} = R_i^{Req} + R_i^{Ack}. \tag{4.9}$$

As mentioned before, the control messages are transmitted using an independent network layer (control layer), similarly to many already commercially available NoCs, e.g., MPPA-256 and Tile64. That is, each processing node has a direct

connection to PANC (cf. Figure 2.4). Thus, the worst-case transmission time of any control messages is the only time required to send such a message via the link (i.e., the maximum link traversal time).

Definition 11. *The worst-case processing time R_i^{Proc} a task i may experience by PANC can be bounded by:*

$$R_i^{Proc} = C_k^{Busy} + R_i^{Blk} + C_i^{Comp}, \tag{4.10}$$

where C_k^{Busy} denotes the worst-case busy time PANC requires to process a task k, which is being processed while a task i arrives; R_i^{Blk} denotes the worst-case blocking time (Blk) the task i may experience by the higher priority tasks; and C_i^{Comp} denotes the worst-case computation ($Comp$) time a task i may experience by PANC to be served.

PANC employs Static Priority Non Preemptive (SPNP) scheduling policy to arbitrate between tasks. That means a task i might wait for the time the PANC requires to complete processing of task k, which is being processed while a task i arrives [106]. Furthermore, in the worst-case, we consider the critical instant scenario when all senders are activated simultaneously, and they are issuing the consecutive requests with the maximal rate. That means while PANC is processing a request, other requests are waiting in its input buffer. Thus, the worst-case processing time of a task i is highly dependent on the concurrent higher priority active tasks. In order to calculate R_i^{Proc}, let N denote the set of senders assuming that the sender does not send another request unless the first one has been served; and let $C_{s,i}$ denote the computation time it takes PANC to process a task i within one stage (cf. Figure 3.11). Then the individual factors of the Equation 4.10 can be calculated as follows:

$$C_k^{Busy} = (C_{s_1,k} + C_{s_2,k} + C_{s_3,k}), \forall k \in N \tag{4.11}$$

$$R_i^{Blk} = \sum_{j \in hp(i)} \left\lceil \frac{R_i^{Blk} + J_j^R}{T_j} \right\rceil \cdot (C_{s_1,j} + C_{s_2,j} + C_{s_3,j}) \tag{4.12}$$

$$C_i^{Comp} = (C_{s_1,i} + C_{s_2,i} + C_{s_3,i}), \tag{4.13}$$

where $(C_{s_1,k} + C_{s_2,k} + C_{s_3,k})$ denotes the computation time it takes PANC within its three stages $(C_{s_1}..C_{s_3})$ to complete processing a current task k; the blocking time (R_i^{Blk}) of a task i corresponds to tasks that have higher priority than i ($hp(i)$) – considering their multiple occurrences $\left\lceil \frac{R_i^{Blk}+J_j^R}{T_j} \right\rceil$ during the time interval $[0, R_i^{Blk}]$.

Definition 12. *The worst-case delay R_i^{Del} a task i may experience by PANC induced by Power-Gating scheme in case the routers are turned off can be bounded by:*

$$R_i^{Del} = R_{PG\_OFF} + T_{WU} + BET, \tag{4.14}$$

where R_{PG_Off} denotes the worst-case time required to propagate a power-gating-off signal that retrieves the power supply to the corresponding router; T_{WU} denotes the router wake-up latency; and BET, as described before, denotes the Break-Even Time necessary to overcome the Power-Gating energy overhead.

The interpretation can be derived using the following worst-case scenario. When PANC tackles a request from a task i, we assume that one or many routers along the packet's path are turned *off*. Then, the latency overhead of turning one router on is the same of turning-on multiple routers simultaneously because of parallelism provided by the control layer (direct connections between PANC and routers). Moreover, we consider in the analysis that the powered-off routers violate the *Hop-Count* slack, and hence the task should account for the wake-up latency T_{WU}. *BET* factor also counts as we assume in the worst-case that the router is just powered-off and right next cycle another request arrives.

Furthermore, we have included the delay policy overhead in the PANC latency analysis (the Equation 4.8) in order to demonstrate the worst-case latency overhead the packet may experience employing *PG* by PANC. However, to make PANC much more efficient as it checks the potential delay at runtime (as introduced in Section 3.3.1), we excluded the delay overhead (induced by *BET*) from the checking of the safe applicability of PANC at design time. That is, the considered worst-case latency overhead induced by PANC is limited to the following equation:

$$MLO_{PANC} = R_i^{Ctrl} + R_i^{Proc} + T_{WU}. \tag{4.15}$$

Finally, to assure timing guarantees at design time, the constraint $DSlack_i > 0$ must be satisfied for each critical sender. Note that known relative deadline for each critical task is an essential requirement in real-time systems [148, 106]. Otherwise, meeting deadlines cannot be formally guaranteed. We remind that upon negative slacks ($DSlack_i < 0$) the application of PANC must be excluded from routers along the packet's path of the corresponding violated task. In other words, the corresponding routers are always *on*, and the violated tasks do not synchronize their NoC accesses with PANC – utilizing a direct access.

PG Analysis Extension under Scalability

We have demonstrated the scalability of the control layer for deployments in many-core NoC-based systems (see Section 2.2.5). In the following, we present

an initial view of the temporal analysis of PANC employing *PG* under scalability. Regarding the additional latency overhead induced by the communication between PANCs through the control NoC, the worst-case latency overhead of PANCs employing *PG*, introduced in Equation 4.8, is modified as follows:

$$\begin{aligned} MLO^{Del}_{PANCs} = {} & R^{Ctrl}_i + R^{LPANC}_i + R^{LPANCDel}_i \\ & + 2 \cdot R^{Trans}_i \cdot NbrRPANCs \\ & + \max_{RPANC \in \mathscr{S}_{PANCs}} (R^{RPANC}_i) \\ & + \max_{RPANCDel \in \mathscr{S}_{PANCs}} (R^{RPANCDel}_i), \end{aligned} \tag{4.16}$$

where R^{LPANC}_i: denotes the worst-case processing time required by local PANC $(LPANC)$ to process a request;
$R^{LPANCDel}_i$: denotes the worst-case delay a task i may experience by a local PANC, induced by the Power-Gating scheme in case the local routers are turned off;
R^{Trans}_i: denotes the worst-case transmission time required by the control NoC to transfer a request from/to the local PANC from/to the remote one;
$max(R^{RPANC}_i)$: denotes the maximum worst-case processing time from the whole set of remote PANCs $(\mathscr{S}_{PANCs})$ to process a request forwarded from the local PANC;
$max(R^{RPANCDel}_i)$: denotes the maximum worst-case delay from the whole set of remote PANCs, induced by the Power-Gating scheme in case the remote routers are turned off;
R^{Ctrl}_i: same as in Equation 4.9.

As discussed earlier, both R^{LPANC}_i and R^{RPANC}_i correspond to the processing time required by PANC to process a request (cf. Equation 4.10). Both $R^{LPANCDel}_i$ and $R^{RPANCDel}_i$ correspond to the worst-case delay induced by the *PG* in case the local/remote routers are turned off.

Furthermore, in case of multiple regions, PANC must create a request and send it through the control NoC to remote PANCs. Thus, R^{Trans}_i is needed to analyse the control NoC delay. We can use any standard NoC analysis frameworks, e.g. [168, 155, 107], to derive the worst-case transmission latency overhead required by the control NoC. However, we employed in this work the holistic analysis [79, 123] introduced in Section 4.1.1. The R^{Trans}_i factor is multiplied by two, one corresponds to the request transmission from the local PANC to the remote one, and the second one corresponds to the acknowledgement transmission from the remote PANC to the local one. Moreover, to account for multiple remote PANCs, the transmission factor is also multiplied by the number of remote PANCs $(NbrRPANCs)$.

DVFS and Mode Change Analysis

This section bounds the worst-case response time of a task on NoC when PANC features the dynamic voltage and frequency scaling scheme [98, 90]. The *DVFS* introduces an additional latency overhead at every change of NoC energy mode, induced by adjusting NoC frequency by PANC as described in Section 3.3.2.

Definition 13. *The worst-case response time $R_i^{PANC(DVFS)}$ of an activated instance of a task i, accounting for PANC employing DVFS can be bounded by the following formula:*

$$R_i^{PANC(DVFS)} = B^{req} + R_i^{MCL_{PANC}}, \tag{4.17}$$

where B^{req} denotes the maximum time to acquire the NoC access, and $R_i^{MCL_{PANC}}$ the worst-case traversal time accounting for additional interference through PANC. We don't need to account for the overhead of releasing the NoC for the task i, as this happens after the transmission.

Each EM defines the worst-case latency for running transmissions while accounting for the maximum possible interference in the NoC. During the transitions between EMs, some tasks can be stopped, new tasks can be activated or, in case there are multiple processing resources (i.e., processors), some tasks can be migrated. In addition, there can be tasks which are present in multiple modes. Consequently, if the transition between EMs happens without the supervision from the PANC, the packets from transmissions initiated in different EMs may interfere. This can cause sporadic overloads and transient load effects, which may endanger the system guarantees. These effects are amplified when applying *DVFS* at runtime. Therefore, the safe switching between different frequency levels, i.e. safe mode changes, has the fundamental importance for the real-time properties of the system and must be safely enforced by the proposed control layer. The mode changes are controlled by PANCs which permit the verification of a NoC with the *DVFS* necessary for certification, not covered by the related research (see Section 3.1). In the worst-case, we assume a request from task i implies a NoC mode change. Thus, to bound the NoC access latency B^{req} , we need to account for sending a request R_i^{req}, stopping active senders ($\max_{j\in\mathscr{S}}\{B^{stopsenders}\}$), receiving an Ack from the stopped senders $R^{ack\_stoppedsenders}$, wait until the initiated packets leave the NoC, scaling the frequency $B^{Freqscaling}$, resuming the senders $B^{resumesenders}$, and finally sending the grant R_i^{gnt}. Thus, B^{req} can be derived as:

$$\begin{aligned} B^{req} =& R_i^{req} + \max_{j\in\mathscr{S}}\{B^{stopsenders}\} + R^{ack\_stoppedsenders} \\ &+ \max_{j\in\mathscr{S}}\{R_j\} + B^{Freqscaling} + B^{resumesenders} + R_i^{gnt}, \end{aligned} \tag{4.18}$$

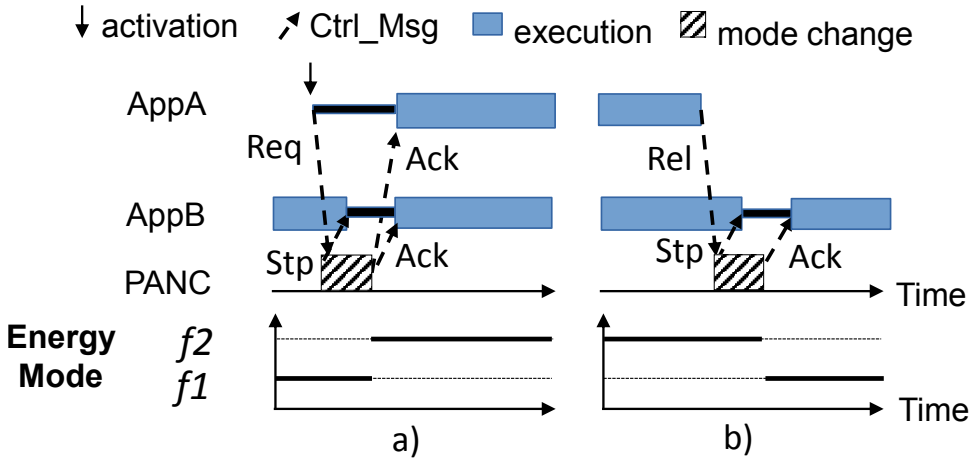


Fig. 4.2.: Protocol workflow for the safe switching between different frequencies in the system: a) increase and b) decrease

where $\max_{j\in\mathscr{S}}\{R_j\}$ accounts for the maximum time needed for the packets of the (stopped) senders $\mathscr{S}$ to leave the NoC. Additionally, the maximum latency of all control messages (R_i^{req}, R_i^{gnt}) is the only time required to send such a message via the link (i.e., the maximum link traversal time). That is mainly attributed to the direct connections between the clients and PANC (cf. Figure 2.4). $\max_{j\in\mathscr{S}}\{B^{stopsenders}\}$ corresponds to the longest latency of the *Stp_Msgs* sent to all senders as well as the time they need to stop themselves. Again, due to the direct connections, the latency of the *Stp_Msgs* becomes the time required to send one *Stp_Msg* via the longest link. The same goes for the *Ack_Msgs* from the stopped senders $R^{ack\_stoppedsenders}$, and resume messages from PANC to the stopped senders $B^{resumesenders}$. Moreover, $B^{Freqscaling}$ corresponds to the maximum time needed for enforcing the new frequency via a clock generator.

Now we need to derive the worst-case traversal time accounting for additional interference through PANC ($R_i^{MCL_{PANC}}$). The mode change in a system can be triggered by two factors: (i) activation of a task (cf. Figure 4.2a), and (ii) termination of an active task (cf. Figure 4.2b). Consequently, we extend the Equation 4.2 as follows:

$$R_i^{MCL_{PANC}} = C_i + \sum_{\phi_j\in\mathscr{F}_i} \left\lceil \frac{R_i^{MCL_{PANC}} + J_j^R + J_j^I}{T_j} \right\rceil \cdot (C_j + B_j) + 2 \cdot \sum_{\forall j\in K} \left\lceil \frac{R_i^{MCL_{PANC}} + J_j^R}{T_j} \right\rceil \cdot B^{ovd}, \tag{4.19}$$

where B^{ovd} denotes the protocol overhead of PANC for handling the requests of other streams.

In the equation, the first line denotes the time needed to transmit the message including the interference from other streams as for the classic *Ri* in Equation 4.2. The second line derives the interference resulting from other requests sent to PANC, e.g., other streams requesting access to the NoC, which leads to an adaptation of the frequency, while a message of task i is being transferred via NoC. For this, we assume in the worst-case that each supervised task causes two mode changes per activation (i.e., activation and termination), and thus the second line is multiplied by 2. The total number of mode changes a task i may suffer depends directly on the total number of the activations of other senders ($\left\lceil \frac{R_i^{MCL_{PANC}} + J_j^R}{T_j} \right\rceil$), where j belongs to the set of tasks that might get activated/released during the task i active time. At every mode change, we need to account for PANC protocol overhead B^{ovd}, which can be upper bounded as follows:

$$B^{ovd} = \max_{j \in \mathscr{S}} \{B^{stopsenders}\} + R^{ack\_stoppedsenders} + \max_{j \in \mathscr{S}} \{R_j\} + B^{Freqscaling} + B_i^{resume}. \tag{4.20}$$

As previously described, in order to avoid sporadic overload, PANC waits until all initiated traffic from all active senders S to safely leave the NoC ($\max_{j \in \mathscr{S}} \{R_j\}$), before changing the frequency ($B^{Freqscaling}$) and resuming the task i (B_i^{resume}), as partially presented in Equation 4.18.

The proposed latency analysis of PANC employing *DVFS* ($R_i^{PANC(DVFS)}$) is slightly pessimistic but guarantees a safe upper bound and allows fairly easy implementation. However, this pessimism can be accepted as the mode change happens only once per full on core activation of a task i. That is, the task synchronises its NoC access with PANC only once for the entire transmission, not per initiated packet, allowed by employing the AER task-model as introduced in Section 2.2.3. Note that we also consider the latency analysis of DVFS under scalability as introduced later in Section 4.1.2.

Integrated Energy Control Analysis

PANC also applies a joint application of diverse energy-savings schemes as presented in Section 3.3.3. This section extends the analysis framework (Section 4.1.1) to upper bound the latency overhead of PANC employing *(PG+CG+DVFS)* scheme [90, 85].

Definition 14. *The worst-case response time a sender* i *needs to conduct a single transmission accounting for the PANC employing the integrated energy control*

$R_i^{PANC(Integ)}$ *can be bounded by the following formula:*

$$R_i^{PANC(Integ)} = B^{req*} + R_i^{MCL^*_{PANC}}, \tag{4.21}$$

where B^{req*} denotes the maximum time to acquire the NoC access including as well the overhead of *PG*/*CG*, and $R_i^{MCL^*_{PANC}}$ the worst-case traversal time accounting for additional interference through PANC.

The time to acquire the NoC access B^{req*} employing *(PG+CG+DVFS)* depends on the current system state. Following the description in Section 3.3.3 and Figure 3.17, there are different possibilities (and thus delays). For the worst-case, we assume the longest of these to occur, i.e., integrating all schemes together *(CG+PG+DVFS)* and thus *Off&FreqScaling* case where the routers are turned off, and the frequency scaling is required. In this case, we need to account for the DVFS timing overhead including stopping active senders $B^{stopsenders}$, receiving an ack from the stopped senders $R^{ack\_stoppedsenders}$, wait until the initiated packets leave the NoC, scaling the frequency $B^{Freqscaling}$, resuming the senders $B^{resumesenders}$; and as well the *PG*/*CG* overhead including sending power/clock-gating on signals R_{PCG_ON} to the respective powered off routers, accounting for their wake-up latency T_{WU}; and surely the overhead of sending a request R_i^{req} and the respective grant R_i^{gnt}. Hence, B^{req*} extends the Equation 4.18 as follows:

$$\begin{aligned} B^{req*} =& R_i^{req} + \max_{j \in \mathscr{S}} \{B^{stopsenders}\} + R^{ack\_stoppedsenders} \\ &+ \max_{j \in \mathscr{S}} \{R_j\} + B^{Freqscaling} + B^{resumesenders} \\ &+ \mathbf{R}_{PCG\_ON} + \mathbf{T}_{WU} + R_i^{gnt}. \end{aligned} \tag{4.22}$$

Note that PANC monitors the real-time properties at runtime. That is, it checks the available slacks and decides whether it is allowed to delay a task by *BET* cycles (as introduced in Section 4.1.2). Therefore, we exclude, at design time, the delay overhead induced by *BET* from the worst-case time the task requires to access NoC (B^{req*}).

The worst-case traversal time accounting for additional interference through PANC ($R_i^{MCL^*_{PANC}}$) is quite similar to the Equation 4.19, except that we extend the busy window under which we analyse the interference from other requests. As such a delay might already delay stream i while it is not actively sending flits but rather suffering from, e.g., the *PG*/*CG* latency overhead, we use the overall transmission time $R_i^{PANC(Integ)}$ when deriving the number of interfering requests.

Consequently, the Equation 4.19 becomes as follows:

$$R_i^{MCL^*_{PANC}} = C_i + \sum_{\phi_j \in \mathscr{F}_i} \left\lceil \frac{R_i^{MCL^*_{PANC}} + J_j^R + J_j^I}{T_j} \right\rceil \cdot (C_j + B_j) + 2 \cdot \sum_{\forall j \in K} \left\lceil \frac{\mathbf{R_i^{PANC(Integ)}} + J_j^R}{T_j} \right\rceil \cdot B^{ovd}. \quad (4.23)$$

Eventually, to assure temporal guarantees at design time, the deadline slack that defines the time difference between the task deadline and $R_i^{PANC(Integ)}$ must be positive $(DSlack_i > 0)$ for each critical sender.

In the following, additional necessary extensions to the analysis framework of the control layer are presented to comprise much larger NoCs, where multiple PANCs are employed.

Integral Analysis Extension under Scalability

When larger NoCs are considered, we first split a NoC into subsets called regions, then specialize one PANC to one NoC region, as introduced in Section 2.2.5. Thus, Figure 4.3 illustrates the extension of the control layer architecture targeting, e.g., 16×16 NoC, and adopting the interconnection between *PANCs*. PANCs are connected through an additional control layer transferring only control messages between PANCs. The control NoC, in our example, is composed of 4 switches (2×2 NoC).

Now we illustrate the comprehensive communication protocol between PANCs, when *(PG+CG+DVFS)* scheme is considered, and present an initial view of the respective extension of the timing analysis framework. Overall, once PANC receives a request/release message from a task in its local region, it checks whether the active task is local or remote upon its number (considering that the task includes its number in the request message). In other words, it checks in its database whether the task is labelled as local (same region destination), or remote (remote region destination). In case of remote access, PANC forwards the request to the corresponding remote PANC/PANCs in case the transmission comprises multiple regions. All involved PANCs follow the workflow in Figures 3.17/3.18 to respectively tackle the request/release message. With *DVFS*, PANC investigates the frequency scaling, and frequency islands between regions must be considered. That in turn requires augmentation of asynchronous buffers as interfaces between multi-frequency regions as is in prior voltage-frequency islands schemes [82, 37, 95]. To reach to a compromise between less area overhead (fewer buffers) and coarse

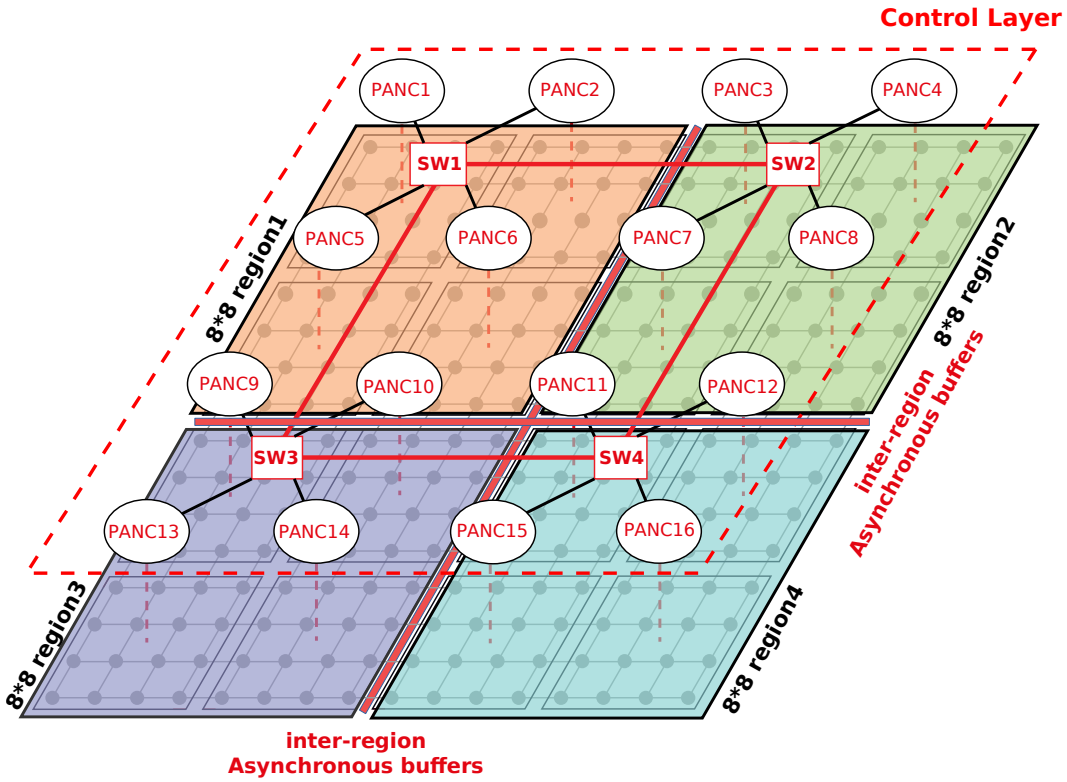


Fig. 4.3.: The extension of the proposed control layer with 16×16 NoC, employing *(PG+CG+DVFS)* scheme

energy-savings granularity, we assume that tasks, which are frequently communicated with each other are mapped right next to each other. As a consequence, each 8×8 region often conducts in-region transmissions (i.e., inside 8×8 region), and rarely inter-region transmissions (between 8×8 regions). Based on our assumptions, each 8×8 region runs under one frequency. However, different 8×8 regions might adopt multi-frequency domains, and thus buffers usage (red bars between 8×8 regions in Figure 4.3) is indispensable.

Thus, regarding in-region communication, the requester PANC distinguishes between two cases. In case frequency increase is required, it notifies other PANCs (within the same 8×8 region) with the new frequency, and waits for their *Ack_Msgs*. Afterwards, PANCs enforce the new frequency. On the other hand, a negotiation process is triggered in case of frequency decrease. To tackle this, PANC negotiates the lower frequency with other PANCs within the same region. It enforces the new energy mode only if all PANCs are tolerant with it (still meets their tasks' deadlines), otherwise, it selects the lowest possible frequency. When using multiple PANCs, there is an additional latency overhead induced by the communication between PANCs and the regions' reconfiguration. This influences the time needed to process a request (B^{req*} in Equation 4.22) and PANC protocol overhead induced by other streams (B^{ovd} in Equation 4.20). We consider

in the worst-case a frequency scaling is permissible, and thus we need to extend Equation 4.22 as follows:

$$\begin{aligned} B^{req*} =& R_i^{req} + \max_{P\in\mathscr{P}_i}\{R_P^{req}\} + \max_{P\in\mathscr{P}_i}\{R_P^{gnt}\} + R_i^{gnt} \\ &+ \max_{r\in\mathscr{R}_i}\Big\{ \max_{j\in\mathscr{S}^r}\{B^{stopsenders}\} + R_r^{ack\_stoppedsenders} \\ &+ \max_{j\in\mathscr{S}^r}\{R_j\} + B_r^{Freqscaling} + B_r^{resumesenders} \\ &+ R_r^{PCG\_ON} + T_r^{WU}\Big\}, \end{aligned} \tag{4.24}$$

where $\mathscr{P}_i$ and $\mathscr{R}_i$ respectively denote the set of other involved PANCs and all concerned regions. The first line now accounts for the time to forward the request to the other PANCs and PANCs sending back the acknowledgement. For this, we obtain the maximum latency for transmitting a message between PANCs in PANC control network. Additionally, the other lines of the equation (i.e., the *max*-function starting at line 2) account for the reconfiguration overhead in each region. As the reconfiguration happens in parallel in all regions, we need to account for the slowest region, covered by the *max*-function. Furthermore, we also need to adapt the $R_i^{MCL^*_{PANC}}$ from Equation 4.23 to account for the additional PANCs overheads as follows:

$$\begin{aligned} R_i^{MCL^*_{PANC}} =& C_i + \sum_{\phi_j\in\mathscr{F}_i}\left\lceil \frac{R_i^{MCL^*_{PANC}} + J_j^R + J_j^I}{T_j} \right\rceil \cdot (C_j + B_j) \\ &+ 2\cdot \sum_{\mathbf{j}\in\mathscr{S}^{\mathbf{other}}}\left(\left\lceil \frac{R_i^{PANC(Integ)}}{T_j} \right\rceil \times B^{ovd*} \right), \end{aligned} \tag{4.25}$$

where $\mathscr{S}^{\mathbf{other}}$ denotes the set of interfering streams from the same and other involved 4×4 regions, and B^{ovd*} denotes the PANC protocol overhead induced by other local and remote streams. To compute B^{ovd*}, we need to account for all streams (from all concerned 4×4 regions) that may interfere with stream i as follows:

$$\begin{aligned} B^{ovd*} =& \max_{P\in\mathscr{P}_i}\{R_P^{req}\} + \max_{P\in\mathscr{P}_i}\{R_P^{gnt}\} \\ &+ \max_{r\in\mathscr{R}_i}\Big\{ \max_{j\in\mathscr{S}^r}\{B^{stopsenders}\} + R_r^{ack\_stoppedsenders} \\ &+ \max_{j\in\mathscr{S}^r}\{R_j\} + B_r^{Freqscaling}\Big\} + B_i^{resume}. \end{aligned} \tag{4.26}$$

The equation accounts for the messages between PANCs (first line), then assumes the maximum reconfiguration time from all affected regions, and finally resumes the suspended stream i.

Moreover, in case of transmissions comprise multiple 8×8 regions, PANC only informs the remote PANC (in the adjacent 8×8 region) with the new request/release message. Then, both of them investigate the aforementioned in-region communication protocol to explore the new situation with their local PANCs. In case of request message, the requester PANC waits for an *Ack_Msg* from the remote ones, which indicate their readiness for the new stream, and finally, upon the delivery of all required *Ack_Msgs*, PANC grants the sender.

4.2. Experimental Evaluation

This section illustrates the efficiency of the introduced control layer through experimental evaluations employing synthetic workload, benchmarks as well as realistic usecases from automotive and avionic domains. To this end, we first present the evaluation methodology and the simulation model (Section 4.2.1). Next, Section 4.2.2 derives the worst-case temporal results of hard real-time tasks integrating the control layer using the introduced analysis framework (cf. Section 4.1) under varying NoC loads. Consequently, the slack budgets of transmissions are computed and validated against the tasks deadlines. Finally, upon positive slacks, we conduct simulations to derive the energy-savings figures for hard real-time NoCs (Section 4.2.2).

4.2.1. Experimental Setup

As networks-on-chip have been emerged as a new on-chip interconnect to efficiently integrate multi- and many-core systems on a single chip, diverse tools have been developed by several research and commercial efforts to provide NoC simulation environments. During the tool design, multiple NoC parameters must be considered such as size, topology, routing, switching, congestion control, link bandwidth, and the buffer depth (number of queues), etc. Some of the available NoC simulators are OMNeT++ that provides HNOCS library [17], BookSim [84], Gem5 that supports Garnet2.0 [4], other libraries from SystemC framework like Noxsim [33] and Nostrum [109]. The implementation of the introduced control layer can be performed in any of the aforementioned NoC simulators as it is mainly independent of the underlying NoC architecture. In this work, we have implemented the control layer using the OMNeT++ simulation framework

due to its flexibility, availability of libraries, code re-usability, popularity among research and industrial communities. OMNeT++ is an event-based, modular, object-oriented NoC simulator. HNOCS has been significantly extended to implement various energy-savings schemes. For more details about the OMNeT++ features and properties, the reader can consult the Section 4.1.2 of the PhD thesis of kostrzewa [97].

Moreover, the number of the considered VCs is equal to the maximum number of congestions for any port, which is in our experiments 2 or 4 VCs. The number of VCs is usecase and analysis framework dependent. More recent analysis framework proposes NoC arbitration schemes that reduce the number of VCs, such as [123], or the use of a resource manager [100] that could be combined with our PANC function. Nevertheless, our approach is not limited to a low channel number. Increasing the number of VCs will definitely increase the absolute router energy overhead. However, regarding energy-savings results, they are mainly affected by the NoC load, as is the case of all energy-savings approaches. In our experiments, at most four streams are sharing a router output port and thus four VCs are sufficient to even avoid head-of-line blocking. Note that higher number of VCs will not reduce the feasibility of PANC as we turn the whole router off once it is idle. Moreover, we use a standard ASIC design flow in order to evaluate PANC area and power overhead. The results are obtained using the VHDL implementation of the IDAMC platform [153] for ASICs [89], where NoC routers are synthesized, placed and routed. Different process technologies from UMC ($65nm$) and TSMC ($28nm$) with core cell libraries of both high and low threshold voltages in worst-case corners (WC, 0.9V, 125°C) and (WC, 0.72V, 125°C) were selected, respectively. After place and route design, the parasitic extraction was performed. The results are back annotated into the designs to allow accurate power measurements.

Moreover, due to the significant advantages of Clock-Gating where the impact of clock-tree power is drastically reduced [89, 115], automatic clock-gating was enabled during synthesis. However, as introduced in [89], the activity on the clock pins of high-level non-clock gated synchronous elements still leads to a clock-tree power overhead. Thus, the latter is tackled using the introduced *(CG+PG)* scheme. Besides, we use the aforementioned predictable tasks accesses model (AER), and therefore we employ Direct Memory Access (DMA) engines in order to adopt the long transmissions. Also, 4×4 and 8×8 2D mesh NoCs are employed in the experiments. A router is composed of 5 ports, with 4 ports connecting to neighboring routers and the fifth one connecting to a tile (e.g., processor or memory). The router power dissipation $P(V_i, f_i)$, considered in this work, can be

computed as follows:

$$P(V_i, f_i) = P_{buffer} + P_{switch} + P_{link}, \tag{4.27}$$

where P_{buffer}, P_{switch}, and P_{link} respectively correspond to the power dissipated at input buffers, switch, and link. Table 5.2b details the leakage, switching and clock-tree power dissipations for the router model transferring one packet. The NoC frequency ranges between [100MHz - 1.8GHz]. From 100MHz to 1.1GHz, the worst-case corner libraries (WC, 0.9V, 125°C) and (WC, 1.08V, 125°C) were used. This means we conducted multiple experiments on different NoC netlists that result from the usage of different ASIC libraries. Since the maximum frequency supported by worst-case libraries is 1.1GHz under *65nm* technology [89], the high performance libraries (i.e., leakage libraries) from 1.2GHz up to 1.8GHz were also used in order to use frequencies higher than 1.1GHz in our experiments. Afterwards, these power numbers at router level are employed in our simulation environment to calculate the energy footprint of the whole NoC by considering the definition of total energy $E(V_i, f_i)$ as follows:

$$E(V_i, f_i) = P(V_i, f_i) \cdot \Delta t \tag{4.28}$$

In addition, concerning *PG*, we considered the *BET* is 10 cycles and the wake-up latency is 2 cycles, consistently with prior research [38, 112, 44]. The *BET* value depends on various parameters, such as sizes of a power switch and a decoupling capacitance. The authors in [77] provide *BET* values based on typical parameters on a typical microprocessor, which has been modified by [112] to estimate *BET* on an on-chip router. Regarding *PG*, as a considerable amount of energy is spent for router on/off switches, we considered in our experiments that each switch is compensated by additional router *BET* active time. Thus, the effective sleeping time is considered after accounting for *BET* that compensates the aforementioned energy loss. Note that more details about the simulation setup are summarized in Tables 5.2a.

Table 4.1.: Simulation Setup

(a) Key Simulation Parameters

Network topology	2D Mesh
Network size	4×4, 8×8 NoCs
Routing algorithm	XY source routing
Switching technique	wormhole switching
Arbitration	Static-Priority Preemptive (SPP)
Link bandwidth	35bits/cycle, consistently with [153]
Flit size	140 bits
Router pipeline, ports	4-stage, 5-port
Input buffer depth	2,4 VCs, 5 flits/VC
Technology	65nm, 28nm

(b) Router power footprints for ASICs under 65nm

	Frequency (MHz)	**Voltage (V)**	**Lkg (mW)**	**SW (mW)**	**clock-tree (mW)**
0.9V, WC	**100**	**0.9**	0.0146	0.1603	0.865
	300	0.9	0.0146	0.4538	2.910
	400	0.9	0.0146	0.61	3.9
	500	0.9	0.0146	0.7516	4.841
	700	0.9	0.0146	1.038	6.774
1.08V,WC	**800**	**1.08**	0.024	1.710	11.2
	900	1.08	0.024	1.897	12.6
	950	1.08	0.024	1.976	13.275
	1100	1.08	0.024	2.213	15.3
1.2V, Lkg	**1200**	**1.2**	0.583	2.780	19.0
	1300	1.2	0.583	2.91	20.1
	1400	1.2	0.583	3.161	22.2
	1600	1.2	0.583	3.442	24.9
	1800	1.2	0.583	3.779	28.0

4.2.2. Simulation-Based Energy-Savings

In this section, we use the aforementioned evaluation methodology (cf. Section 4.2.1) to evaluate NoC energy-savings through the control layer using PANC. Diverse energy-savings schemes, *PG*, *DVFS*, *PG+CG*, and *PG+CG+DVFS* are employed. We demonstrate the PANC efficient results employing periodic pattern from different workloads: realistic usecases from automotive and avionic domains, real benchmark suite, and synthetic workloads.

Realistic Usecase: Automotive Domain

We first employ a realistic usecase from automotive domain extracted from the work of Shi et al. [145] to demonstrate the PANC impact on power and performance using a real application. In this section, PANC features Power-Gating and integrated energy control (*PG*, *CG*, and *DVFS*). For these experiments, we employ 4×4 NoC, 4 VCs, and 500MHz clock frequency. The chosen application is the control of a vehicle with assistance functions, as it considers both large flows as well as control loops with short messages. The vehicle is designed to recognize an unknown space by populating a database of obstacles, obtained by stereo photogrammetry and ultrasonic sensors.

Power-Gating

In this work, several flow periods and data sizes from the employed usecase have been changed to increase the throughput of the application' tasks. Task periods vary between 0.4 ms to 1 ms, and communication volumes vary between 9 kB and 0.3 MB. The workflow graph and the corresponding mapping are illustrated in Figure 4.4 where the router R_5 (highlighted in red) is the tracked one in the experiments. The functionality selected for experiments is composed of 18 communicating tasks where all transmissions are performed periodically. Furthermore, we employ the aforementioned formal analysis 4.1.2 on the selected usecase, and the slacks of hard real-time tasks are extracted. Figure 4.5 plots the worst-case response time of the application' tasks employing PANC ($R_i^{PANC(PG)}$, see equation 4.7) along with their corresponding deadlines. As it can been seen, the applicability of PANC is feasible due to high available slacks.

Figure 4.6 details the power consumption of the tracked router in non-Power-Gating (No-PG) scheme (no Power-Gating is applied), and PANC along with its optimization. We break down the router power into dynamic power (switching power), leakage power, clock-tree power, and Power-Gating power overhead which is mainly induced by power cycling (turning routers on/off) and PANC

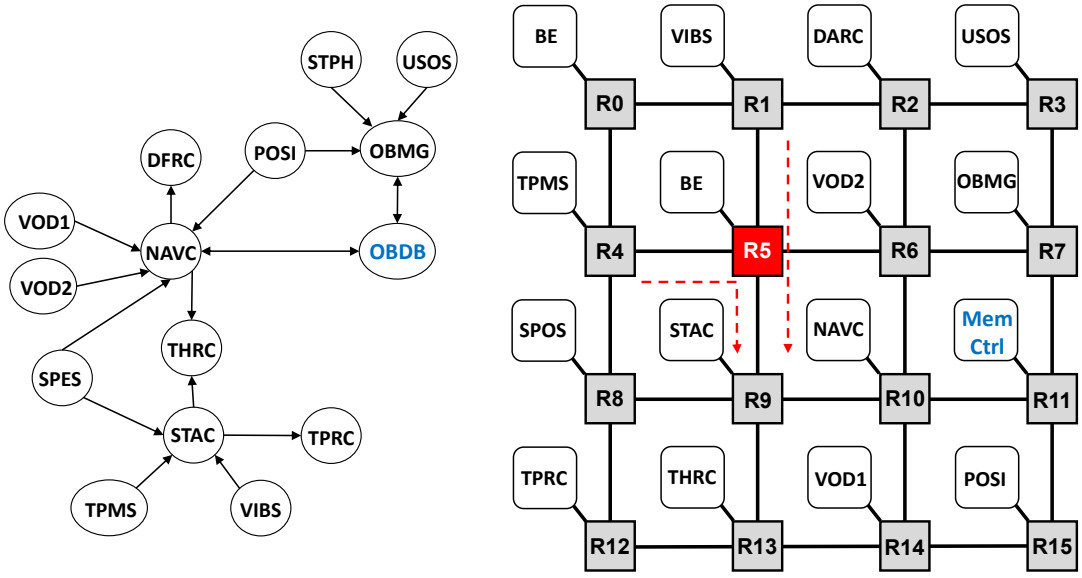


Fig. 4.4.: Graph (left) and mapping (right) of selected tasks from the autonomous vehicle use-case [145]

unit. To be fair, we refer to the static power as the total of clock-tree, leakage, and even *PG* overhead. In this experiment, the power cycling overhead is very low (not-shown). This arises from the fact that real application loads typically have low injection rates (the tracked router load is 7.2%).

As Figure 4.6 shows, PANC has significant impact on static power-savings compared with No-PG scheme under different technology libraries 65*nm* and 28*nm*. It saves up to 93% of the static power (a factor of 12 better). Results conducted by 28*nm* technology show the efficiency of PANC even at higher leakage power consumption. The efficiency of static power-savings employing PANC is mainly achieved by its global controlling and then decision making. Thus, it turns off routers directly when all tasks are *off*, and accounts for *BET* rule when it must turn routers on. Moreover, as it is anticipated, the optimized PANC (Opt-PANC) has almost the same impact on power as basic PANC with very slight improvement (not shown). This is attributed to the low data rates with large volume flows in the used usecase. Consequently, the router experiences very few number of switches, thus enhancing PANC's functionality by optimizing it to decrease the switches number does not have a considerable impact on power. However, in agreement to the explanation introduced in Section 3.3.1, higher impact of Opt-PANC arises at higher data rates with short messages (shown in Section 4.2.2).

Moreover, in order to better illustrate the global insight of our approach across the whole NoC, Table 4.3 depicts the relative power-savings of all routers in 4×4 NoC under PANC, using 65*nm* technology. While some of routers (highlighted in red) have to stay *on* for longer time due to high congestion/preempted tasks

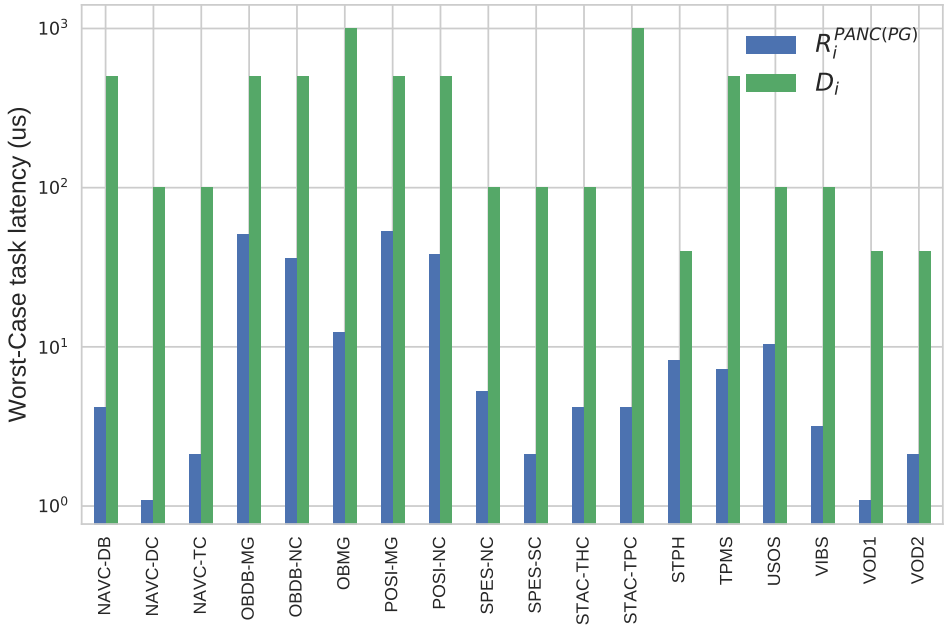


Fig. 4.5.: Analysis-based worst-case task latency of the vehicle usecase [145]

Table 4.3.: Power-savings of individual routers in the 4×4 NoC, employing the autonomous vehicle usecase [144]

(y,x)	0	1	2	3
0	90.3%	91.08%	99.37%	78.67%
1	95.90%	92.59%	68.39%	77.85%
2	79.9%	79.08%	62.29%	70.5%
3	99.17	97.94%	73.30%	66.82%

with reserved channels, others can be turned off earlier – fulfilling higher power-savings. On average the network power-savings using our approach is 82.7% compared with No-PG. To be fair, we demonstrate as well the power overhead of the introduced control layer (PANC, additional links, and clients). The experimental results, using 65*nm*, indicate that the power overhead of PANC induces only 0.84% increase of the basic NoC power consumption. Moreover, the links (6-bit wide each in case of *PG*) and clients induce 0.44% and 0.048% increase of the basic NoC power, respectively. After accounting for the power overhead of the control layer, the average NoC power-savings using our approach become 81.4% compared with No-PG. More details about the PANC implementation and its overhead are introduced later in Section 4.2.3.

Regarding performance, the simulation results indicate that the absolute increase of the average packet latency employing our approach compared with No-PG is

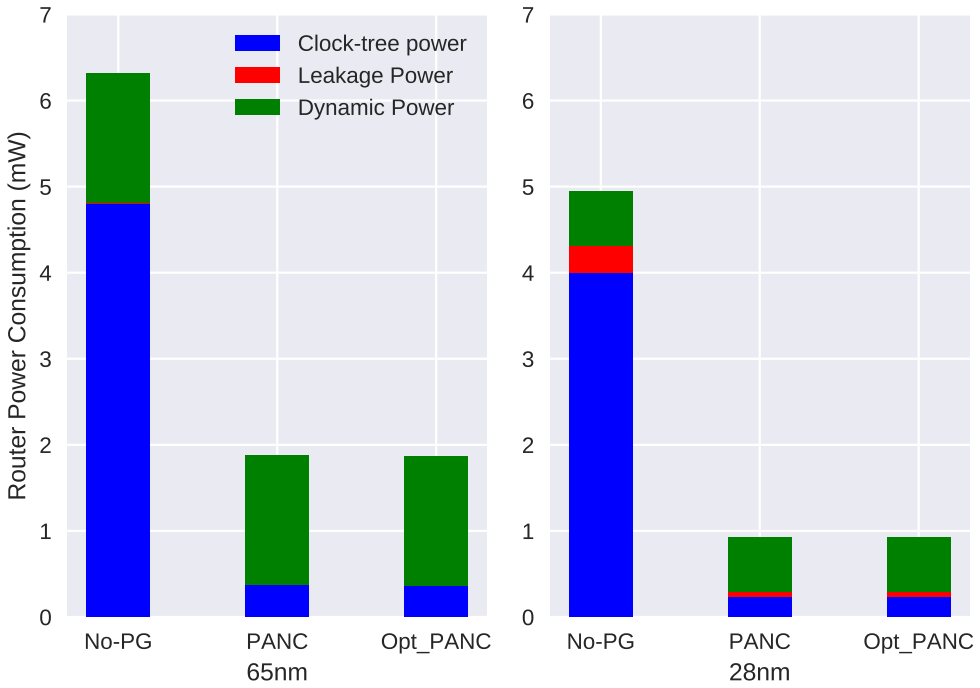


Fig. 4.6.: Router breakdown power of the vehicle usecase under 65*nm* and 28*nm* technology libraries

8 cycles. Hence, the percentage increase of the average packet latency is minor in case of big message sizes, which already demand huge number of cycles to be transmitted via NoC.

Integral Evaluation

In this section, using the aforementioned realistic usecase, we evaluate the better efficiency of PANC employing the joint application of the energy-savings schemes, *PG+CG* and *PG+CG+DVFS*, over the individual ones. Several flow periods and data sizes from the employed usecase have been changed to increase the throughput of the application' tasks, and thus illustrate better PANC different aspects. Task periods vary between 0.06 ms to 1 ms, and communication volumes vary between 9 kB and 0.43 MB. The workflow graph and the corresponding mapping are demonstrated in Figure 4.7. The functionality selected for experiments is composed of 18 communicating tasks where all transmissions are performed periodically. Furthermore, we employ first the aforementioned formal analysis (cf. Section 4.1.2) on the selected usecase in order to derive the safe applicability of PANC. Second, we break down the energy consumption of both router and NoC levels employing PANC with the following energy-savings schemes:

- ***PG:*** PANC features only *PG*, and thus targets leakage energy-savings.
- ***PG+CG:*** In order to reveal the impact of PANC on both leakage and clock-tree

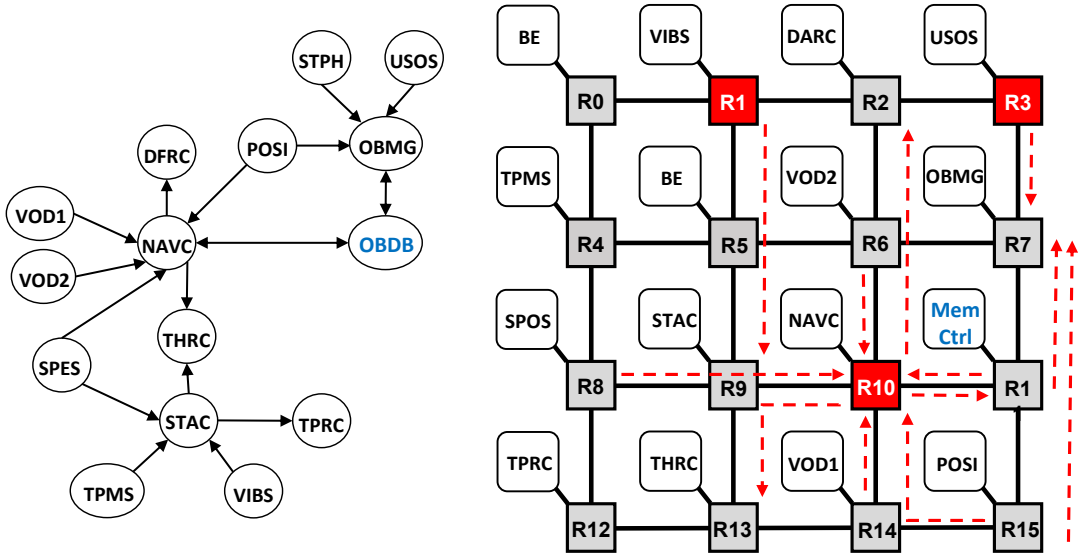


Fig. 4.7.: Graph (left) and mapping (right) of selected tasks from the vehicle usecase [145]

energy sources, PANC combines both *CG* and *PG*.

- ***DVFS:*** In this case, PANC features only *DVFS* and thus targets as well the switching energy-savings.
- ***PG+CG+DVFS:*** In this scheme, PANC combines the functionality of *PG*, *CG*, and *DVFS* to explore the potential additional energy-savings.

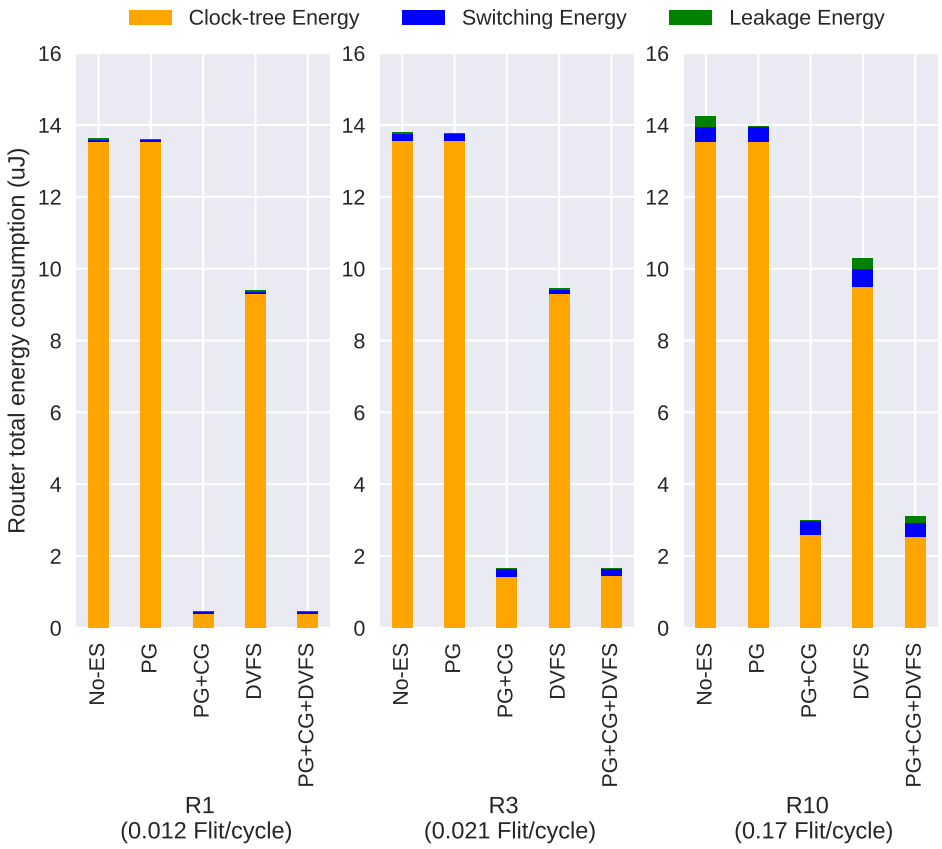
Note that we refer by ***No-ES*** to the baseline NoC where none of the energy-savings schemes has been applied.

Router level Figure 4.8a details the energy consumption employing PANC along with its multiple energy-savings schemes compared with *No-ES*. The energy consumption of three routers $\{R_1, R_3, R_{10}\}$ at different loads $\{0.012, 0.021, 0.17\}$ flit/-cycle has been plotted under 65*nm* technology, respectively. We break down the router energy into switching, leakage, clock-tree sources, and power cycling energy overhead induced by Power-Gating scheme. We refer to *PG* overhead as part of clock-tree and leakage. In these experiments, the power cycling overhead is trivial (not-shown). As it is depicted in Figure 4.8a, *PG* achieves almost zero energy-savings in total compared with *No-ES*. That is mainly attributed to the fact that *PG* targets only a reduction of leakage energy that constitutes a small portion of the router total energy. Regarding *DVFS*, it shows better savings footprint than *PG* in all cases. That is fulfilled by mitigating the clock-tree and switching energy consumptions at lower frequencies. Moreover, employing *PG+CG* scheme is quite decent as gating the complete impact of clock-tree once a router is powered off implies significant clock-tree energy-savings. Regarding R_3, although the router's load is not much higher than R_1, we see the energy-savings are less.

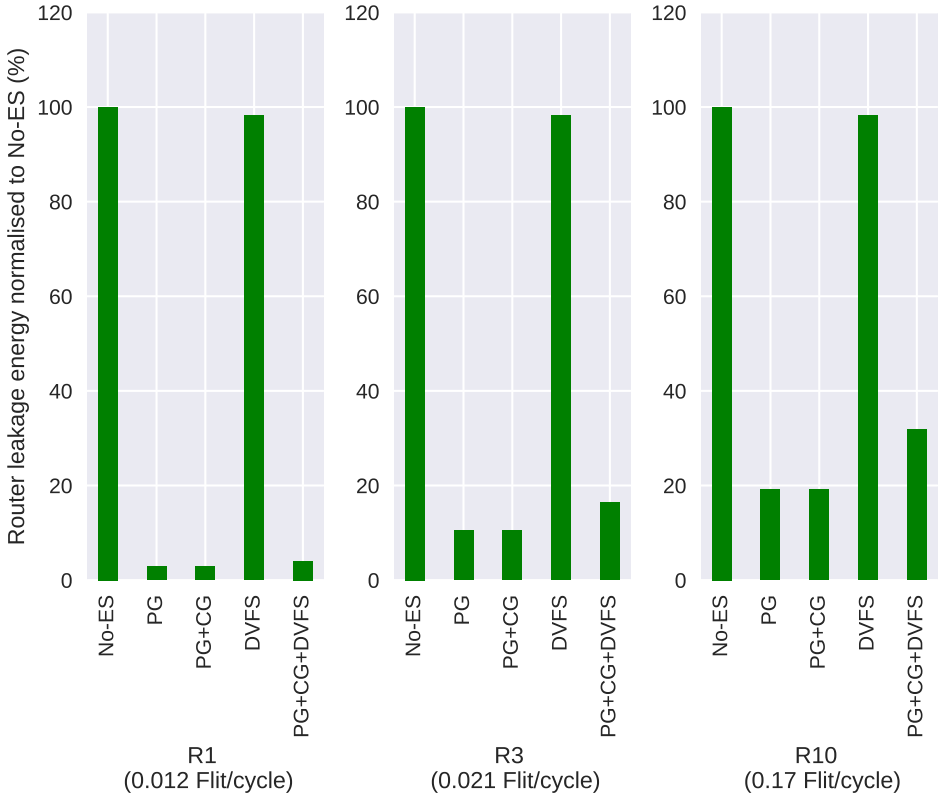
That is a result of the congestion at R_7 caused by interference from higher priority streams, which in turn keeps R_3 *on* – doing nothing.

In addition, at low load (R_1), employing *PG+CG+DVFS* scheme through PANC has mainly as energy-savings impact as *PG+CG*. Despite the fact that *PG+CG+DVFS* mitigates the energy consumption of all energy sources, the impact of *DVFS* on decreasing the clock-tree energy by scaling the frequency down does not significantly appear. That is attributed to the fact that frequently scaling the NoC frequency up/down, makes the NoC slower at low frequencies, resulting in fewer opportunities to powering a router off than *PG+CG*.

Regarding leakage energy, as it is too small and thus is not obvious in Figure 4.8a, Figure 4.8b plots the routers' leakage energy under different schemes relatively to the *No-ES*. *PG* safely and efficiently saves leakage energy by powering the router off once it is dormant. As it is depicted, it depends on the router load (R_{10}) and congestion (R_3). *DVFS* has, in these experiments, zero impact on leakage energy-savings as leakage is dissipated even at zero loads. When PANC targets *PG+CG+DVFS* scheme, it almost achieves the same leakage savings as *PG* at low rates (R_1). However, as previously described, at higher rates/congestions (R_{10}, R_3) powering the router off is performed less frequently employing *PG+CG+DVFS* due to slowing down the NoC at lower frequencies. That even leads to save less leakage energy.



(a) Breakdown of routers total energy



(b) Routers leakage energy

Fig. 4.8.: Routers total (a) and leakage (b) energy of the vehicle usecase employing different functionalities of PANC

NoC level To better emphasize the efficiency of PANC at NoC level employing different energy-savings schemes, additional simulations have been conducted. To the best of our knowledge, this is the first work that investigates integrating multiple energy-savings schemes, while simultaneously providing hard real-time temporal guarantees. Thus, we can only compare with non-integrating solutions.

We compare the integrated energy control with [91] and [98] that respectively apply the *PG* and *DVFS* on real-time NoCs. The total, switching, and leakage energy consumptions of NoC employing the vehicle usecase at different NoC load rates have been explored. Clock-tree energy is implicitly expressed by the total energy as it comprises the majority of total energy. As previously mentioned, in real-time systems finishing a critical task early has the same advantage of completing it by its deadline [148]. Thus, we employ in our experiments the worst-case based results that indicate the temporal safety of all critical tasks. To this end, we employ at each rate the aforementioned formal analysis (cf. Section 4.1) to safely apply PANC. The results indicate that the worst-case response time R_i of some interfering tasks (of the employed application) in *No-ES* (baseline) NoC violates the deadline after 24.2% average requested link bandwidth. The temporal safety has been violated in the baseline NoC, leading obviously to stop applying any energy-savings schemes that in turn increase the timing violation.

We investigated as well the energy overhead of the introduced control layer (PANC, additional links, and clients). The experimental results, using 65*nm*, indicate that the links (7-bit wide each) and all clients respectively induce 0.481% and 0.088% increase of the baseline NoC power consumption. Regarding PANC, the associated quiescent power highly depends on the feature that PANC targets. In case of *PG*, the power overhead of PANC induces 0.85% increase of the basic NoC power. However, *PG+CG* induces more power increase (1.58%). Employing *DVFS*, PANC induces 1.22% and 2.79% increase of baseline NoC power under *DVFS* and *PG+CG+DVFS*, respectively. Thus, we account for the respective energy overhead of the control layer approach in all energy figures in this section.

As previously described, in these experiments we used three different libraries to support various data rates. This results in three different NoC netlists that we used to illustrate the effective of our approach under varying NoC utilizations. Figure 4.9 depicts the NoC total energy consumption under *No-ES*, along with the relative savings of the aforementioned schemes. As it is anticipated, the savings impact of *PG* is very trivial as the leakage constitutes a very small portion of total energy. Its impact appears at high NoC utilizations, and thus high frequencies as the respective voltages increase (cf. Table 4.2b and Figure 4.9c). When combining *CG* to *PG (PG+CG)*, the energy-savings significantly outperform the *PG*. We achieve 91.1% savings compared with *No-ES*. That is mainly attributed to

effectively mitigate the impact of the clock-tree energy that constitutes the major portion of total energy (cf. Figure 4.8a).

Moreover, *DVFS* savings' impact varies along different NoC utilizations. It saves significant energy at low requested link bandwidths, as it runs NoC under the lowest safe frequency. These savings decrease at higher requested bandwidths to start increasing again after 8.1% (cf. Figure 4.9b). The reason is that after 8.1%, higher frequencies are required and thus we used another library to support the higher load rates resulting in another netlist. That in turn requires higher voltages, and thus higher power consumption (cf. Table 4.2b) for *No-ES* scheme. However, *DVFS* still has the opportunity to run lower frequencies under less activated tasks, and thus save better energy.

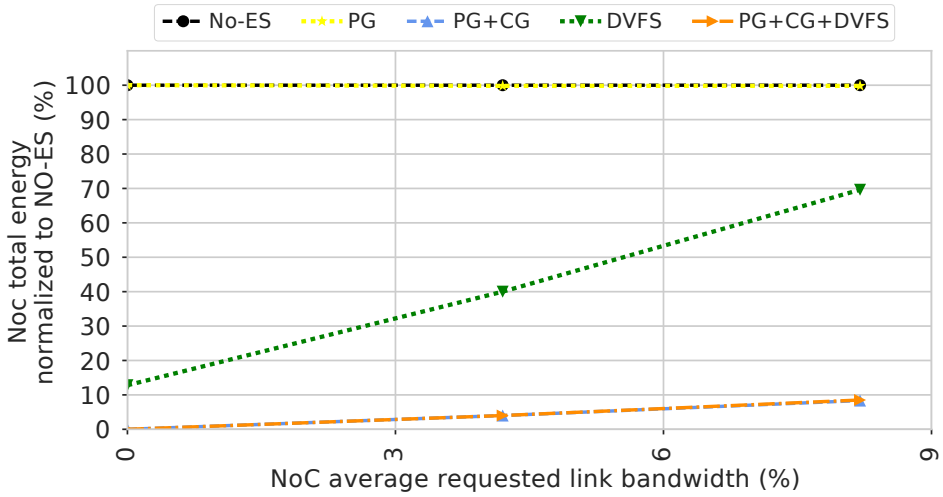
Furthermore, in these experiments, combining *DVFS* to *PG* and *CG* has almost zero better savings than *PG+CG*. The only thing *DVFS* can add, under the combination, is decreasing the energy dissipation during the active time of NoC routers by decreasing the frequency where possible. That in turn highly depends on the simultaneously active tasks. In other words, if most of the tasks are active most of the time, *DVFS* sets a high frequency most of the time – less savings. As previously mentioned, enabling *DVFS* increases the NoC active time (due to the employed lower frequencies) compared with *PG+CG*, increasing the respective clock-tree energy.

Regarding switching energy, Figure 4.10 indicates that *PG+CG+DVFS* has higher impact on switching savings than *PG+CG*. That was not obvious in total energy figures as switching energy constitutes a small portion of the total energy. Furthermore, *DVFS* outperforms all schemes in switching energy-savings. *DVFS* has better savings when NoC utilizations increase (cf. Figures 4.10a and 4.10b). However, at higher utilizations (Figure 4.10c), *DVFS* curve increases again, resulting in less savings.

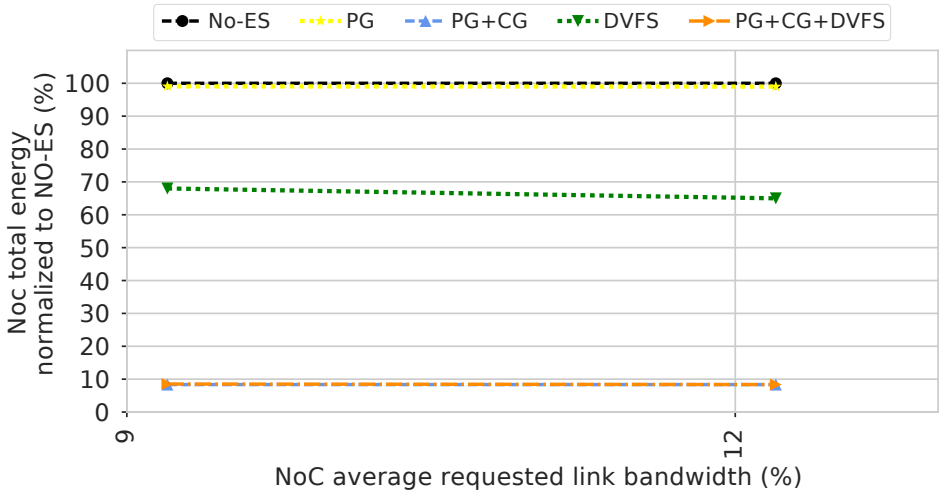
Regarding leakage energy, Figure 4.11 indicates that *DVFS* saves zero energy at low NoC utilizations. However, at higher ones (cf. Figures 4.11b and 4.11c), *DVFS* curve shows considerable leakage savings as scaling down the frequency decreases the higher leakage energy required at higher frequencies (cf. Table 4.2b). As anticipated, *PG* and *PG+CG* schemes dramatically save leakage energy as they mainly switch off the router once it is dormant. By employing *PG+CG+DVFS* scheme, the leakage savings increase at higher NoC utilizations.

In conclusion, since the experimental results indicate that the major dissipation of NoC energy is caused by the clock-tree, configuring PANC to *PG+CG* scheme is quite effective. Moreover, we see the impact of *PG+CG+DVFS* on total energy-savings is almost the same of *PG+CG* under 65*nm* process technology. However, as systems-on-chip become more complex, considering technology downscaling

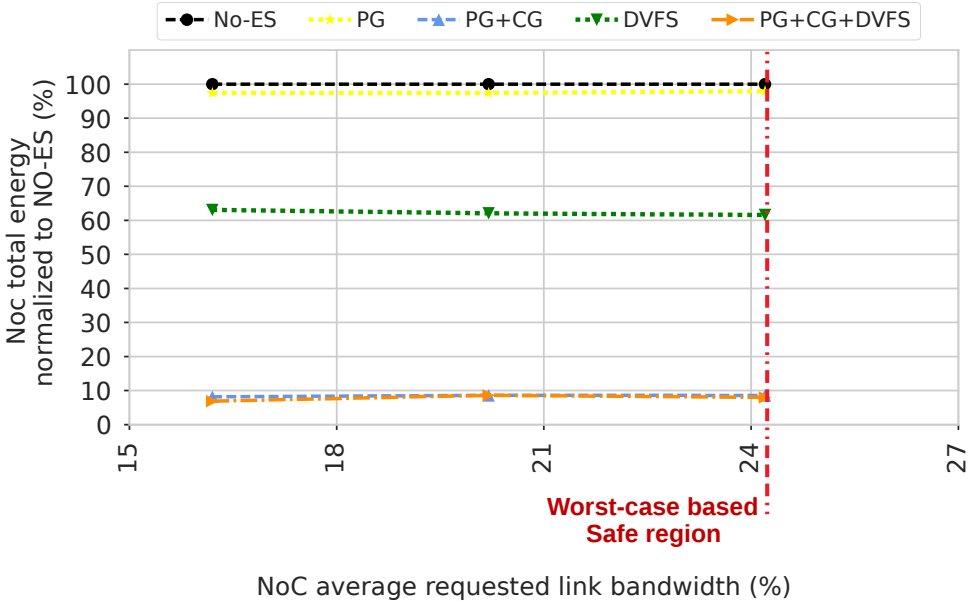
is indispensable. Note that PANC will have higher impact on energy-savings for NoCs with smaller process technologies (e.g., 28*nm*) as the leakage and the clock-tree energy consumptions will be relatively higher [91]. Consequently, combining *DVFS* to *PG+CG* will show better energy-savings footprint.



(a) Library1 (0.9V, WC)

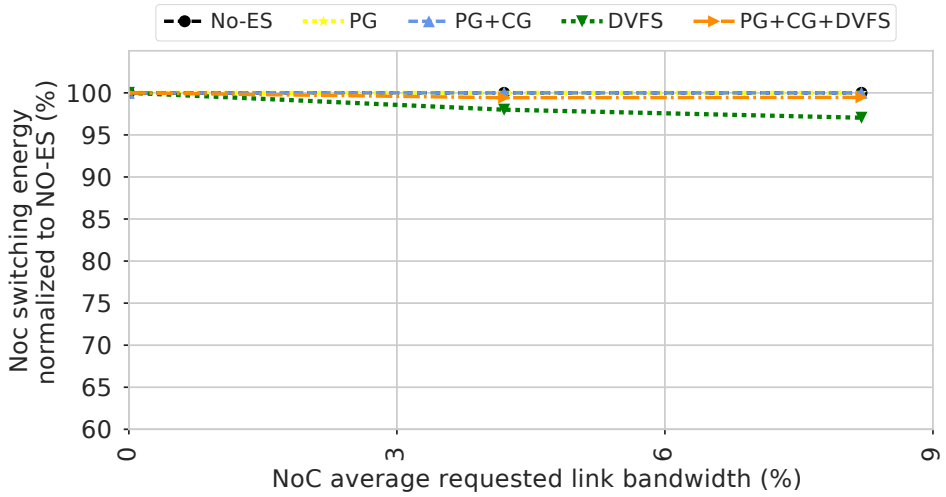


(b) Library2 (1.08V,WC)

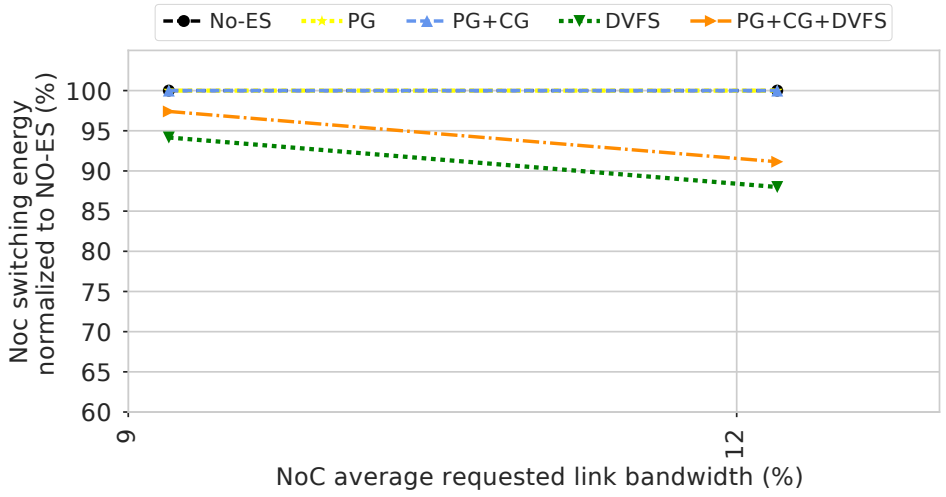


(c) Library3 (1.2V, Lkg)

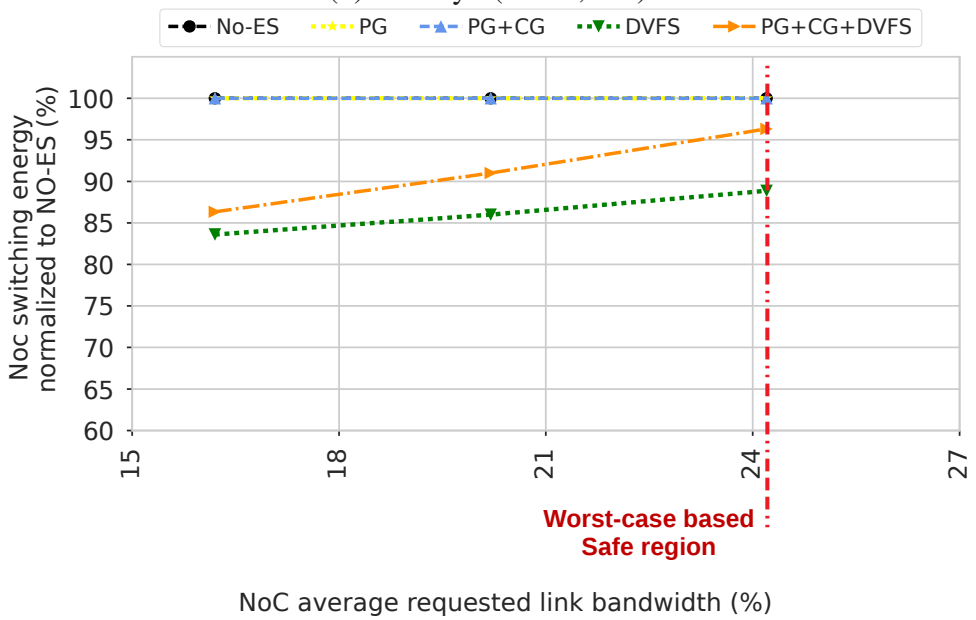
Fig. 4.9.: NoC total energy of the vehicle usecase is normalized to *No-ES* scheme, using three different libraries and employing various energy-savings schemes through PANC



(a) Library1 (0.9V, WC)

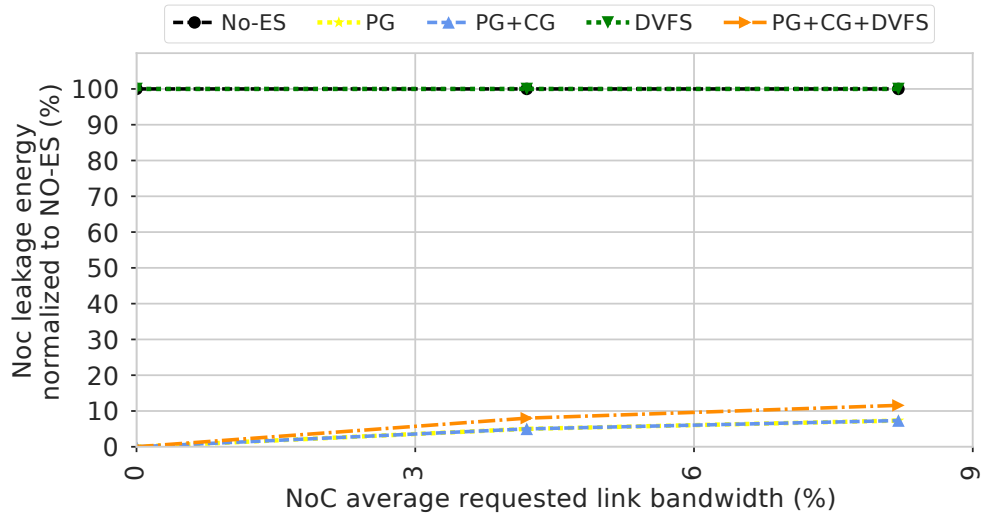


(b) Library2 (1.08V,WC)

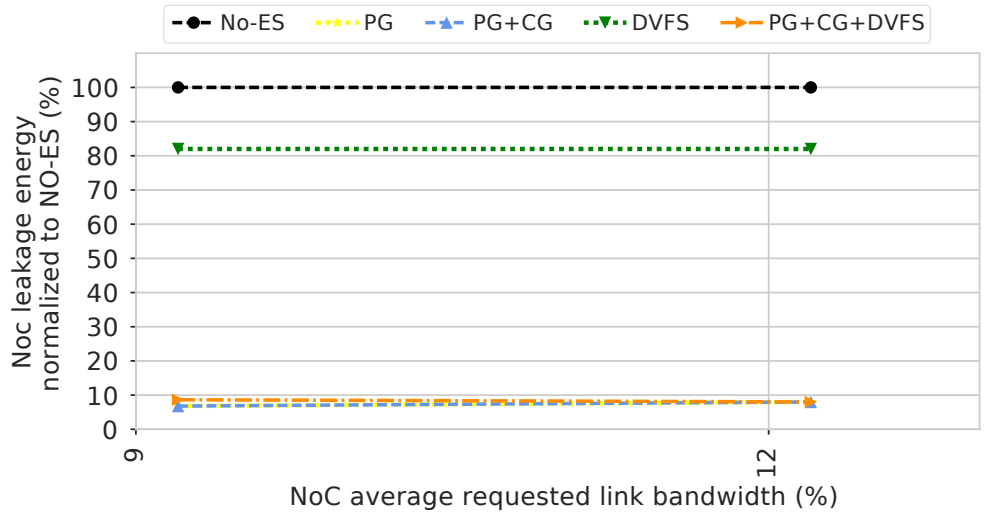


(c) Library3 (1.2V, Lkg)

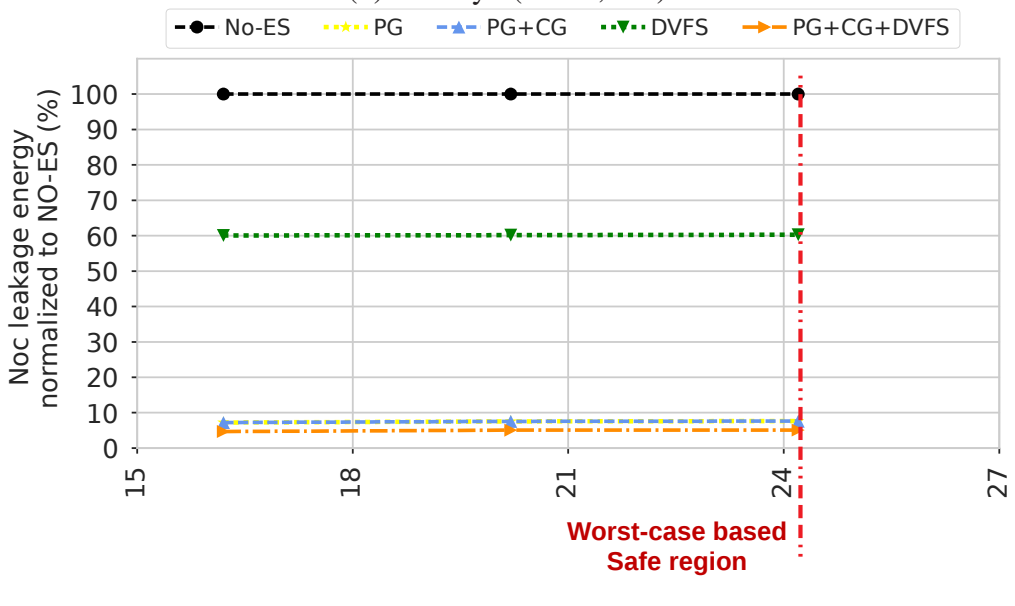
Fig. 4.10.: NoC switching energy of the vehicle usecase is normalized to *No-ES* scheme, using three different libraries and employing various energy-savings schemes through PANC



(a) Library1 (0.9V, WC)



(b) Library2 (1.08V,WC)



(c) Library3 (1.2V, Lkg)

Fig. 4.11.: NoC leakage energy of the vehicle usecase is normalized to *No-ES* scheme, using three different libraries and employing various energy-savings schemes through PANC

Realistic Usecase: Avionic Domain

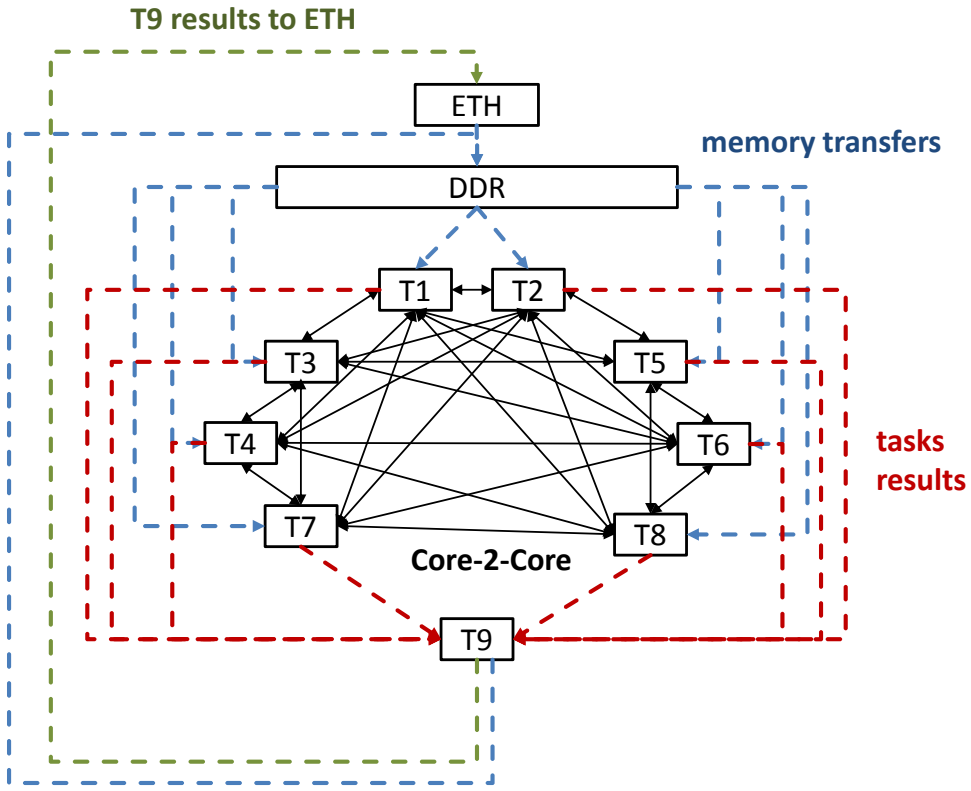
In order to evaluate the scalability of PANCs, we conduct simulations for 8×8 NoC (supervised by 4 PANCs connecting by a switch). Consequently, the NoC is divided into four regions. We consider the usecase from [2], which is from safety-critical domain. The critical applications are modelled based on the Full Authority Digital Engine Control (FADEC) application, see Figure 4.12a. For achieving Triple Modular Redundancy (TMR), FADEC application is executed in three regions in parallel. We consider as well the Health Monitoring (HM) application, which is gathering the results from FADEC instances in order to monitor their functionality in the MPSoC. In case of an erratic behaviour, i.e., receiving different results from FADEC instances, HM outputs an error message via ETH2. Thus, based on our usecase where 4 applications are employed, we use 4 PANCs (one PANC per application).

The considered workflow of FADEC is described in the scope of actions conducted by tasks. First, it receives from an Ethernet interface sensors data from an engine 90 kB. These data are stored into the DDR memory and later distributed to n tasks, noted T1 to Tn. During their parallel execution these tasks, except the last one Tn, exchange data in direct transmissions. Task periods vary between 0.5 ms to 1.2 ms, and communication volumes vary between 2.2 kB and 60 kB. Finally, they provide results to the last task Tn, which stores 30 kB in the DDR memory, and additionally sends back 15 kB through the same Ethernet interface. Figure 4.12a presents the graph of the communication in the FADEC application, assuming 9 tasks. In our example, presented in Figure 4.12b, we scale the communication to reach n = 16 tasks. The fourth region (orange) in Figure 4.12b considers HM application. Note that to avoid high inter-region sporadic traffic due to dependencies between applications, we mapped tasks belong to different regions right next to each other (e.g. the red arrows in Figure 4.12b). Moreover, we assume a MPSoC with two independent single port memory modules (DDR1 and DDR2) as well as four Ethernet ports (ETH1-4). Therefore, FADEC1 uses ETH1 and DDR1, HM uses ETH2 and DDR1, FADEC2 uses ETH3 and DDR2, and finally FADEC3 uses ETH4 and DDR2.

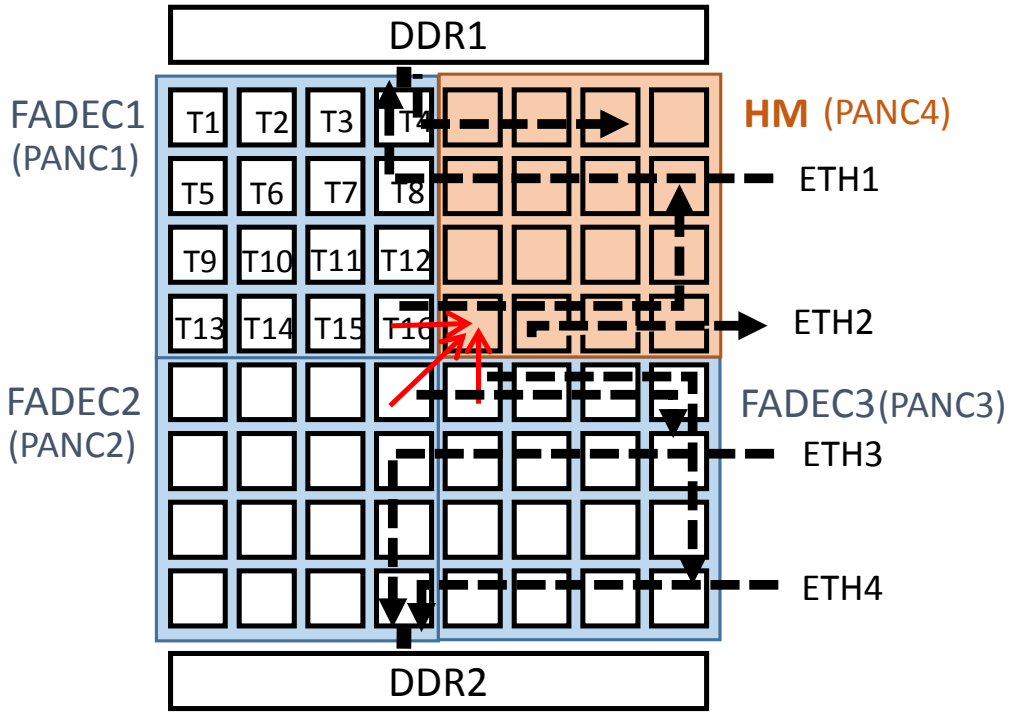
Regarding power-savings, we assured first safe applicability of PANCs (featuring *PG*) employing the formal analysis framework introduced in Section 4.1.2. Next, we run the simulation, using 65*nm* technology, and the experimental results indicate that PANCs save up to 79.13% of the static power compared with No-PG NoC after accounting for the power overhead of the control layer.

In general, we introduced in this chapter the experimental evaluations of the control layer approach employing PANC, which is capable to feature one/combi-

nation of diverse energy-savings schemes. The user is highly flexible to configure PANC to any of the aforementioned schemes that implies different energy-savings granularities. That is highly dependent on the chip's process technology, tasks' behavior, and the application real-time requirements.



(a) Task graph of FADEC application



(b) Mapping of three FADEC applications and one HM application

Fig. 4.12.: Task graph of avionic usecase - FADEC application (a), and mapping of three FADEC applications and one HM application (b) using 8×8 NoC

Benchmark suite: MiBench applications

We employ another realistic workload derived from benchmark applications to evaluate the effectiveness of PANC featuring Power-Gating scheme under different aspects. For this, we use benchmarks from the MiBench suite [66] as real-time applications that is introduced earlier in Section 3.2.1. We recall that we employed two simulation scenarios, *data intensive* with 14.1% NoC load, and *mixed workload* with 5.3% NoC load in 4×4 NoC. In these experiments, we use 4 VCs with the tracked router R_6 which is the closest router to the memory and fully connected (see Figure 3.2).

Regarding power-savings, we assured first safe applicability of our approach employing the formal analysis framework introduced in Section 4.1. Next, Figure 4.13 details the power consumption of the tracked router under 65*nm* process technology. The Figure depicts a comparison between non-Power-Gating (No-PG) scheme, and PANC (featuring *PG*) along with its optimization (cf. Section 3.3.1). Recall, we refer to the static power as the total of clock-tree, leakage, and even power cycling power overhead. To be fair, we also added the PANC power overhead to the router power consumption. Using *data intensive* scenario (14.1% NoC rate), PANC achieves up to 67.96% savings of the router static power, and up to 71.41% employing the Opt-PANC. Furthermore, using the *Mixed workload* (5.3% NoC rate), the router static power-savings are 84.94% and 86.24% under PANC and Opt-PANC, respectively. As it is anticipated, the Opt-PANC has higher impact on power-savings compared with basic PANC. That is mainly attributed to the capabilities of Opt-PANC that decreases the number of router transitions between the on/off states, and thus decreases the Power-Gating power overhead.

Regarding performance, the simulation results indicate that the relative increase of the average packet latency has a small performance penalty. Using *Data intensive*, the average latency increase employing PANC compared with No-PG is 6.3%. Additionally, the average packet latency under *Mixed workload* is only 2.8%.

Synthetic workloads

To better understand the relative aspects of PANC along with its optimization (featuring *PG*), we conduct simulations with synthetic traffic across the full range of network loads until saturation employing 65*nm* process technology. In these experiments, we use uniform random traffic with a 5-flit packet size. We compare PANC with the following four schemes:

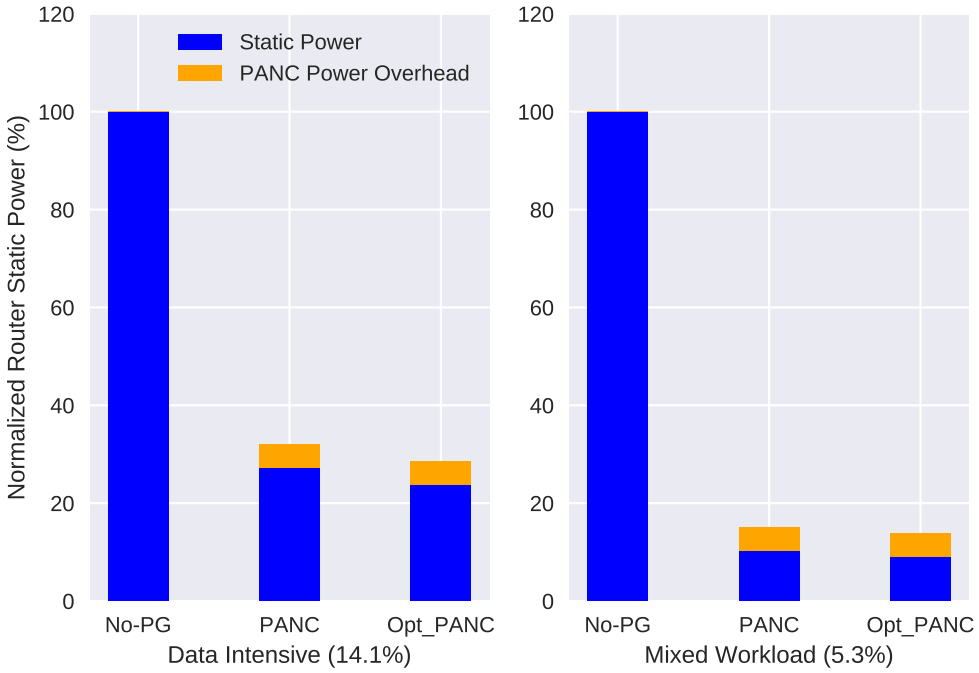


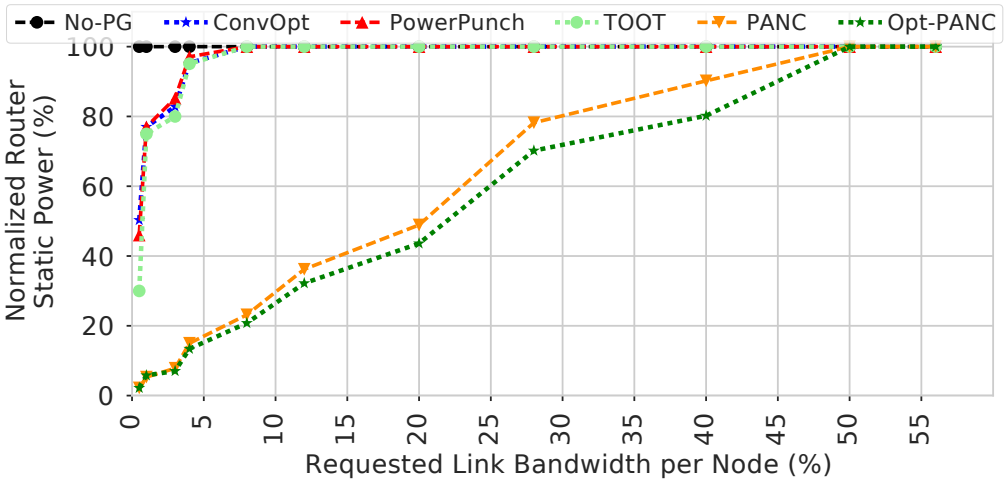
Fig. 4.13.: Router static power normalized to the baseline NoC (No-PG) using MiBench Benchmark, employing *Data intensive* and *Mixed workload* scenarios

- **No-PG:** The baseline NoC.
- **ConvOpt:** Like conventional Power-Gating schemes but it uses the early wake-up signal [112] to partially hide the wake-up latency.
- **Power Punch:** Which efficiently utilizes early power punch signal in order to completely hide routers wake-up latency.
- **TOOT:** Which supports routers with TOOT component in order to increase the router sleep period.

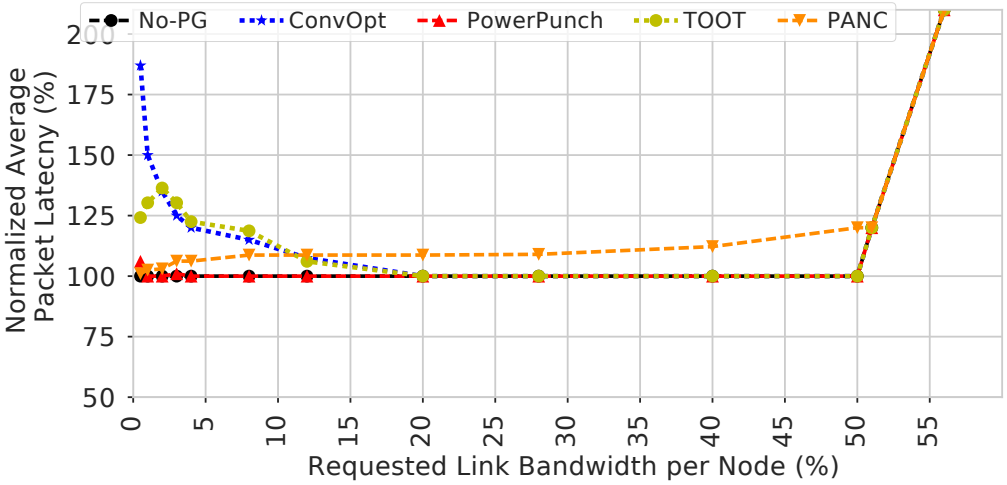
Figure 4.14a compares the router static power results of PANC (featuring *PG*) along with the aforementioned schemes (as reported in [39, 58]), normalized to the No-PG case. Note that the results are reported for the router R_5 (highlighted in red in Figure 4.4), where the power values include Power-Gating overhead and the control layer power consumption. Moreover, in order to illustrate the applicability of PANC under real-time systems and best-effort ones (no deadline constraints), Figure 4.14a plots two regions:

- **Safe region:** PANC is safely applied on the system.
- **Unsafe region:** corresponds to the best-effort case.

Using the aforementioned formal analysis (cf. Section 4.1) under the uniform random setup, the results indicate that the worst-case response time R_i of some



(a) Router static power normalized to No-PG case



(b) Average packet latency normalized to No-PG case

Fig. 4.14.: Router static power (a) and average packet latency (b) normalized to No-PG case using uniform random traffic

interfering tasks in No-PG scheme violates the deadline after 28% requested link bandwidth. The temporal guarantees are already violated in the baseline NoC, leading obviously to stop applying any Power-Gating methods that in turn increase the timing violation. As it is depicted in Figure 4.14a, PANC considerably outperforms other schemes in static power-savings. Moreover, when data rate gradually increases, we observe that PANC advances other schemes by saving power, which stop saving too early (12%). That mainly comes from the fact that other schemes do not efficiently overcome the *BET* violation, i.e., routers are turned on without accounting for *BET* which in turn, at higher rates, dramatically deceases the power-savings. In contrast, PANC, which keeps routers turned-off for an adequate time ($>BET$), decreases considerably the impact of violating *BET*. In other words, PANC power curve slightly increases over the full data range comparing with other curves' sharply increase. On the other hand, in unsafe region,

we also observe that PANC has far better impact on static power-savings than others, especially at rates under which other schemes do not achieve any savings. Moreover, as Figure 4.14a shows, PANC keeps saving power until NoC's saturation, and thus plays a crucial role in power-savings even in best-effort systems at high requested link bandwidth. Note that once PANC stops saving power (e.g. at saturation case), and thus keeps routers *on* all the time, the control layer must be shut down, bringing the NoC back to its normal operation (only data layer). That is mainly in order not to increase the overall power induced by the useless control messages (cf. Table 2.1).

Furthermore, let us focus on Opt-PANC. As it can be seen, Opt-PANC in these experiments has considerable impact on power-savings compared with normal PANC. That is attributed to the fact that optimized PANC not only efficiently overcomes the *BET* challenge but also decreases the number of routers transitions by grouping the task activations. That way, at higher rates with small message sizes, Power-Gating overhead induced by power cycling the NoC routers is considerably decreased. Note that the NoC saturates at 56% requested link bandwidth because of the congestion at output port of the tracked router in the random setup.

Regarding performance, the power-savings advantage of PANC exhibits low performance penalty. To derive the PANC impact on average packet latency and make it comparable with other *PG* schemes, we conduct simulations in OMNeT++ under the uniform random traffic. Figure 4.14b indicates that the increase of the packet average latency under PANC at low rates is negligible. However, at high requested link bandwidth (e.g. 50%), the average latency increases by 20% compared with No-PG case. The latter increase of packet latency using PANC comes from aspects like PANC-Sender synchronization protocol and router states.

On the other hand, Figure 4.14b compares PANC latency overhead with other schemes, as reported in [39, 58]. ConvOpt, which optimizes conventional Power-Gating methods to early wake-up technique, cannot completely hide the wake-up latency. At low requested link bandwidth (e.g. 1%) where many routers are supposed to be *off*, it increases the packet average latency by 50.3% compared with No-PG. Power punch, which efficiently applies early power punch signal, is able to leave only 6.1% increase in packet latency at low rate (0.5%) as reported in [58]). Moreover, Power Punch has almost no impact on performance at higher data rates (e.g. 20%), which is achieved at the cost of less power-savings opportunity comparing to PANC (cf. Figure 4.14a). While the previous schemes target low performance penalty at higher data rates, TOOT, on the other hand, induces higher performance penalty at higher rates because of congestion on TooT's bypass latch. When employing uniform random traffic (2.5% data rate), it increases the packet average latency by 36.4% compared with No-PG, as opposed to our approach that

induces negligible latency overhead at this rate (cf. Figure 4.14b).

4.2.3. Implementation and Resource Overhead

The actual implementation of the control layer units, the PANC and the clients, is highly flexible. In principle, PANC could be implemented as software or hardware. In case of a software implementation, PANC runs on a specified core. PANC performance will be lower, and thus it increases the latency overhead of the approach. However, at the same time, PANC's design is highly flexible allowing easy updates and complex schedules. In case of a hardware implementation, PANC is implemented as an independent hardware module, and thus it introduces additional hardware overhead. Moreover, it induces low flexibility (hard updates) but simultaneously offers high performance (thus allowing high granularity of the protocol). PANC is implemented in this work as a hardware module. As previously mentioned, the evaluation of the area and power overhead of our approach is fulfilled using the IDAMC platform [153] for ASICs [89], employing 65*nm* technology. The implementation results indicate that the size of all on-core hypervisors (clients), required to synchronize the tasks' NoC accesses with PANC when employing *PG+CG+DVFS*, is very small, only 0.102% of *No-ES* NoC area. Similarly, the power overhead is also low, only 0.088 increase of *No-ES* NoC power, as depicted in Table 4.4.

Furthermore, the area overhead of the additional links in the control layer is negligible compared with NoC area. The links (7-bit wide each) induce 0.52% increase of the baseline NoC power. Regarding PANC, the additional area overhead is energy-savings scheme dependent. Thus, targeting only *PG* leads to 0.62% increase of *No-ES* NoC area in case of 4×4 NoC (supervised by one PANC). When combining *CG* to *PG* (*PG+CG*), the increase becomes 0.92%. When *DVFS* is concerned, the area overhead of PANC induces 0.51% and 1.43% area increase under *DVFS* and *PG+CG+DVFS* schemes, respectively. Overall, the control layer increases the NoC size by 1.532%, which is quite acceptable.

Regarding power dissipation, the associated quiescent power highly depends on the energy-savings scheme that PANC targets. In case of *PG*, the power overhead of PANC induces 0.85% increase of the baseline NoC power. However, combining *CG* to *PG* induces more power increase (1.58%). Employing *DVFS*, PANC induces 1.22% and 2.79% increase of baseline NoC power under *DVFS* and *PG+CG+DVFS*, respectively. Hence, the substantial units of the control layer (PANC and clients) have been developed with low hardware and power overheads, satisfying *(Obj3)*, introduced earlier in Section 2.2.1.

Table 4.4.: Area and power overhead of the control layer units: PANC and clients

	PANC				Clients
	PG	*DVFS*	*PG+CG*	***PG+CG+DVFS***	
Area (um^2)	16278	13390	24155	37545	60.84
Relative increase of NoC area (%)	0.62	0.51	0.92	**1.43**	0.102
Power (um)	0.23	0.33	0.43	0.76	0.026
Relative increase of NoC power (%)	0.85	1.22	1.58	**2.79**	0.088

4.3. Summary

The required extensions of NoC design to integrate the introduced control layer forming energy-efficient NoC must be verified and complied with the fundamental requirements of safety-critical hard real-time systems (cf. Section 2.2.1). This chapter presented formal verifications with respect to temporal guarantees of the control layer. The analysis engines have been developed to describe the timing behavior of the control layer primary units (PANCs, clients along with the respective synchronisation protocols). Indeed, transmissions via NoC induced by the control layer must be bounded and the synchronisation protocols as well as PANC functionality must be predicted. To this end, the NoC analysis frameworks from [79, 123] have been extended with multiple mathematical models required to bound the worst-case latency overhead of PANCs featuring divers energy-savings schemes (*PG*, *CG*, *DVFS*). The feasibility of NoC energy optimization employing the control layer has been strongly given via the extracted positive slacks (cf. Section 4.2.2).

Moreover, to evaluate the efficiency of the control layer in terms of energy-savings, various realistic usecases have been utilised, including automotive exemplar (vehicle control), avionic (FADEC), benchmark suite (MiBench), and finally synthetic workload with varying NoC load rates. The evaluation has been performed using ASIC tool chain for both 65nm and 28nm process technology. The experimental results indicate that PANC can achieve 91.1% energy-savings compared with *No-ES* when featuring *PG+CG* scheme (cf. Section 4.2.2). Consequently, the control layer outperforms the status quo so far in NoC energy savings, while satisfying the essential requirements in safety-critical hard real-time

system. First, it adheres to timing constraints via the introduced analysis engines (Sections 4.1), satisfying $Req1$ from Sections 2.2.1. Second, low latency overhead of control messages that is fulfilled through the isolation between the developed control layer and the underlying NoC architecture, fulfilling $Obj1$ and $Obj2$ (see Sections 2.2.1). Third, low resource overhead with respect to power and area. The actual implementation of the control layer along with the respective resource overhead has been demonstrated in Section 4.2.3. Overall, PANCs induce only 2.79% increase of baseline NoC power when employing (*PG*, *CG*, *DVFS*) scheme. In addition, the control layer increases the NoC size by only 1.532%, which is quite tolerable, achieving $Obj3$.

In general, the control layer architecture fulfills the design goals and the deployment requirements in safety-critical hard real-time systems. Hence, an architecture using the control layer for optimizing NoC energy developed on top of an interconnect for data transmissions provides a feasible and productive alternative for the future embedded systems requiring low energy consumption and temporal guarantees.

Chapter 5: Safe Online Adaptations in Mixed-Criticality Systems

This chapter presents the resource management of ICs that implement Mixed-Criticality Systems (MCSs) [88]. As presented earlier, we support the system with hierarchical control layers to treat degradation. Here, we describe the detailed functionality of these layers and their communication protocols. MCSs generally combine safety-critical and non-critical applications on the same execution platform. They are ubiquitous in cyber-physical systems with growing importance as complex function networks are integrated on fewer high-performance computing platforms, such as in automotive platforms for automated driving. Many of those applications require continuous service of critical functions. Continuation requirements can be permanent, such as in a pacemaker or in a smart power grid, or for longer periods of time, such as in industrial robotics, or traffic applications including autonomous driving. MCSs with continuous service require runtime management of the underlying embedded systems platform.

Non-Interference in Mixed-Criticality Systems

Runtime management of MCSs is limited because they are governed by safety standards. More precisely, MCSs have asymmetric requirements, a protected SC part/subsystem that requires functional guarantees, and a BE part that is implemented with optimized efficiency (power and performance), but with no guarantees. This asymmetry could be further refined into several levels of criticality, but the optimization benefit is limited [54], such that we assume a classification into safety-critical and best-effort subsystems. While SC subsystem requires simple runtime environments with sufficiently predictable static behavior to enable thorough verification, the BE subsystem will usually employ a complex OS, such as Linux. The result is a heterogeneous system architecture with fixed boundaries. Boundaries are fixed because safety standards require non-interference of safety-critical functions. Such *freedom from interference* (as in ISO26262 [81]) or *sufficient independence* (as in IEC61508 [78]) is required to permit dynamic

execution profiles with low predictability in the BE subsystem, and thus subject to lower verification requirements in the design process. Virtually, all MCSs exploit that possibility to enable cost efficient high-performance designs with flexible BE subsystems. As a consequence of that asymmetry, the SC applications must not trust functions in the best-effort part. This extends to the runtime environments including runtime resource management [81, 78]. In practice, the boundaries between SC and BE subsystems remain fixed, not only at runtime, but generally after first deployment avoiding expensive re-designs of critical parts.

Towards Flexible MCS Boundaries

There are important reasons to enable flexible boundaries at runtime.

- **Change:** An obvious situation is a change in the execution profile of the SC subsystem requiring more resources (e.g., a software update). If there are no system downtimes, then the two options are moving the boundaries or rejection of change. If none of those are possible, then there must be a fall back system to either continue (fail operational) or stop safely (fail stop). While updates at runtime can usually be rejected, there are other situations where this is more complicated.

- **Error:** The second situation is the occurrence of an error in the SC subsystem. There are many mechanisms in theory and practice to detect and handle such errors, but the fixed boundaries prevent migration exploiting the BE resources due to the non-interference requirement mentioned above. Hence, if the resources in the SC subsystem are insufficient to handle the error, the fall back system is needed.

- **Imminent hazard:** The third situation is the most complex one addressing effects of recent and upcoming IC technologies. Publications in semiconductor physical design [105] state that in many cases degradation is a precursor of failure with many failure mechanisms showing continuous degradation well in advance of failure [105]. While IC degradation effects have been investigated for a decade [70], recent research interest addresses degradation sensing and circuits for online degradation monitoring anticipating failure in SC applications [105, 143]. In this work, we assume the availability of such monitors to detect an increased risk of failure, also called an imminent hazard [135]. In this third situation, the error has not yet occurred and the system is still perfectly operational with all abilities to handle the presented situations (Change and Error). Therefore, any action that would increase risk of failure beyond the status quo must be avoided. That is a hard constraint to any method for handling imminent

hazards.

All three situations have in common that they would profit from flexible MCSs boundaries, where movable boundaries would improve availability over existing mechanisms. Therefore, it is possible to reject a boundary move, but if the boundary is moved, that move must be safe. This requirement includes respecting all SC subsystem guarantees including real-time, where applicable. Although movable boundaries can handle the aforementioned three cases, we focus on developing an approach that is particularly optimized to treat imminent hazard as it is a new phenomenon detectable by [105, 143]. Its use is evaluated on an experimental platform [88]. We assume a many-core system using a network-on-chip that can handle communications independently. Such independent communication is necessary to separate SC from BE traffic. Some commercial examples of multi-/many-core systems are the Kalray MPPA [25] and the Infineon AURIX [96].

Our proposed solution is twofold. First, we propose a hierarchical control approach that provides online reconfigurations, which are particularly beneficial for an imminent hazard handling (foreseen system failures) in many-core platforms. The approach employs hierarchical system-wide cooperation between local and global controllers to perform a workload migration via enabling movable boundaries between safety-critical and best-effort subsystems in mixed-criticality domains. Second, to allow safe online reconfigurations, fundamental requirements including isolation, protected accesses to shared resources, temporal guarantees, and deadlock-free reconfiguration are preserved .

The rest of the chapter is organized as follows. Section 5.1 derives detailed requirements to the development of any mechanism for moving boundaries. After summarizing related work in Section 5.2, we develop our online reconfiguration approach in Section 5.3. Next, Section 5.4 provides a formal analysis framework to reconfiguration timing. Section 5.5 provides experimental evaluations and last Section 5.6 concludes the chapter.

5.1. Key Requirements for Online Adaptations

As previously mentioned, the treatment of imminent hazards requires a system reconfiguration. However, any kind of change must not violate the safety-related aspects concerning the critical functions. That is, any misbehaviour of BE applications must not influence the execution of critical applications (non-interference requirement). It is a challenging requirement as treating imminent hazards by reconfiguration through workload migration between safety-critical and best-effort subsystems/zones affects the required isolation. We remind that the treatment

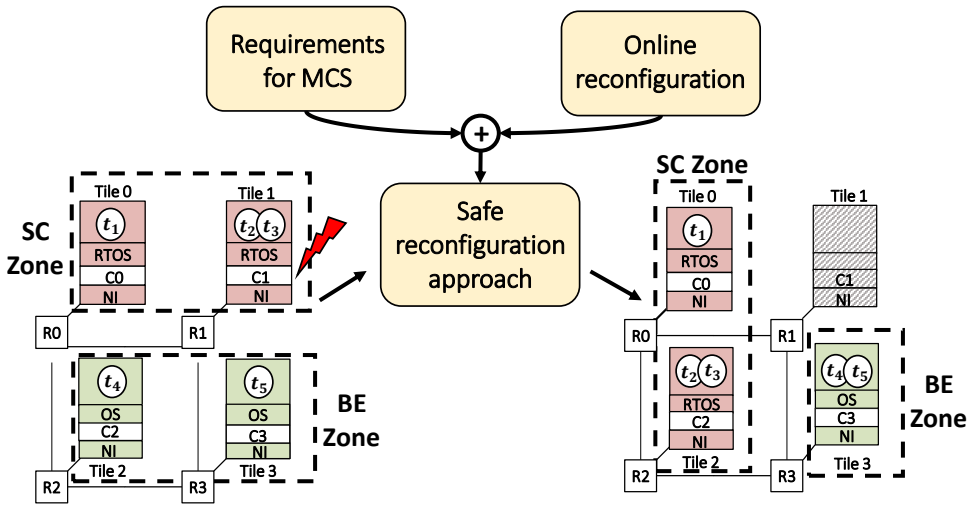


Fig. 5.1.: An introductory example of changing the mixed-criticality subsystem boundaries due to reconfiguration

impacts a properly running system that only possibly may fail later. Thus, risk must not unacceptably be increased by the treatment. We, therefore, treat system reconfiguration with a change of subsystem boundaries and container migration as a safety case, through a safe reconfiguration approach that adheres to the requirements of mixed-criticality systems during reconfiguration (Figure 5.1).

The introductory example in Figure 5.1 consists of many tiles, connected via a shared main memory over a network-on-chip (NoC). The platform is organised in two zones, the SC and BE zones. Once an imminent hazard occurs, the safe reconfiguration approach employs container migration from the tile that expects a foreseen failure $core_1$ (c_1) to an identified BE tile $core_2$ (c_2). This is performed after clearing the BE tile from its current workload, including the OS, which is untrusted to do the migration on its own. Noteworthy, the NI of the identified BE tile becomes also red, repurposing its architectural properties to run the SC container, as detailed in Section 5.3.2.

Based on the aforementioned challenges, we derive requirements to perform safe online container migration, satisfying safety standards in mixed-criticality systems:

- ***Req1: Isolation.*** Always ensure isolation between BE and SC tasks program and data, during and after runtime reconfiguration of the system. That is, never trust the BE part (software not sufficiently qualified against errors). The system must always be able to end in a state where the critical part is properly executing. Thus, isolated reconfiguration of BE and SC zones is required. That also includes protected memory access. Consequently, BE tasks are not allowed to access the memory regions specified only to SC tasks during reconfigura-

tion. Thus, an additional memory protection layer is adopted, as detailed in Section 5.3.

- ***Req2: Protected communication channels.*** The access to NoC resource from BE tasks must be limited and the interference with critical tasks must be bounded not only during regular operation but also during reconfiguration. *Req2* is satisfied through introducing NoC with reconfigurable isolation feature, as detailed in Section 5.3.2.

- ***Req3: Temporal guarantees of timing-critical tasks.*** The reconfiguration imposes an additional delay for those tasks and a deadline miss has a direct impact on the system's functional safety [78, 81, 49]. Thus, the worst-case latency overhead of the reconfiguration must be bounded and validated against the tasks deadlines. That is, all hard deadlines must be met (without allowing a deadline miss), as introduced in Section 5.4.

- ***Req4: Robust reconfiguration.*** The container migration must not bring the system to a deadlock and especially the failure did not occur yet. Thus, the approach must be able to control any erratic behaviour that might be caused by reconfiguration, and ensure the safe ending of the system, as required by the standards [81, 78, 49]. *Req4* is satisfied through employing the principle of the reliable transport protocol, as detailed in Section 5.3.

5.2. Related Work

This section summarizes the relevant state-of-the-art on self-aware and mixed-critical embedded systems. Self-aware computing combines concepts that have been the subject of prior computer science research in areas including artificial intelligence, autonomic computing, self-adaptive and self-organizing systems, and cognitive computing [102]. Mixed-criticality real-time systems are systems that implement functions of two or more distinct criticality levels, such as safety-critical and best-effort [54, 29]. Nowadays, most of the complex embedded systems found, e.g., in the automotive and avionics domains are evolving into mixed-critical systems to increase efficiency and reduce size, weight, and power (SWaP). A comprehensive overview of mixed-criticality is found in [29]. Traditionally, mixed-criticality real-time systems have had static configurations for higher criticalities and flexibility for best-effort functions. There are also systems with room for adaptation [127, 139], which can be classified as self-adaptive systems. Self-awareness in mixed-criticality real-time systems has received relatively

little attention with approaches that focus on a specific aspect, such as [101, 141].

An error has different effects on different layers of hardware and software [10]. If not appropriately detected and handled, errors propagate and can cause system failure [72]. In mixed-criticality systems, addressing hardware errors in time (i.e., before a failure occurs) is crucial. Employing system reconfiguration with task migration to overcome permanent errors and adapt to changes in the system is a recurrent idea. A comprehensive overview of trends in mapping on many-core systems is found in [146]. The authors in [139] propose a hybrid methodology, where a design-space exploration is combined with an agent-based runtime adaptation, which targets dynamic cross-layer reliability. Also, in [114] the authors investigate the model-based approach to allow systems to autonomously adapt their configurations. The authors in [52, 62] introduce the migration mechanisms that replicate the migrating tasks on every local memory. A challenge arises, however, when migrating mixed-critical workload since temporal guarantees must be given to the safety-critical component of the workload not only before and after, but also during the migration process. The authors in [146, 139, 114, 52, 62] do not consider mixed-criticality systems requirements (presented in Section 5.1), nor provide temporal guarantees of hard real-time tasks (violating *Req3*). Furthermore, the authors in [59, 51] present the multi-agent approach for a reconfigurable avionic system, employing mixed-criticality platform. However, the authors do not consider the safety standards requirements during reconfiguration (cf. Section 5.1). Moreover, the work allows a deadline miss which is forbidden in safety-critical hard real-time systems, where analytical proofs showing tolerant reconfiguration latency overhead w.r.t. hard timing constraints are not supported.

The concept of predictable runtime migration is introduced in [116, 122], however, the work does not experimentally demonstrate hard real-time guarantees. In [127, 42] the authors tackle the aforementioned limitation by ensuring temporal guarantees during reconfiguration. The authors in [127] introduce the timely reconfiguration of critical tasks' mapping for NoC-based real-time systems. They rely on a design-space exploration to create a set of possible configurations for the system and a hybrid latency analysis for the migration, which combines a computation intensive discovery of migration routes at design-time with a final latency calculation at runtime. Despite the authors in [127, 42] assure temporal constraints (satisfying *Req3*), they do not address other requirements in mixed-criticality systems *(Req1, Req2, and Req4)*.

In contrast, our approach pursues runtime adaptations in mixed-criticality systems through enabling movable boundaries between SC and BE subsystems, while satisfying the presented reconfiguration requirements (cf. Section 5.1). The functionality of our approach is instantiated as an extension of an existing NoC con-

troller, the power manager (PM) [91, 90, 85, 98]. The PM performs safe NoC power management, besides performance efficiency [101]. Our approach extends the concept of the PM from the NoC level to the system level in order not only to provide energy management, but also safe runtime adaptation to foreseen core failures.

To the best of our knowledge, this is the first work that safely investigates moving the boundaries between SC and BE subsystems at runtime, while preserving fundamental isolation requirements in mixed-criticality systems (cf. Section 5.1). Although the approach can be applied to tackle the introduced two cases, change (e.g., a software update) and error, in the following we elaborate the approach to particularly feature an Imminent Hazard (IH) handling.

5.3. Hierarchical Control Approach

We introduce hierarchical control layers through cooperation between the system controller (global controller) and local controllers to enforce update configurations in the face of imminent hazards. The hierarchical/distributed controllers allow scalability in many-core platforms. The local controllers are, CTCs, and BECs. The proactive handling takes place upon the detection of an imminent hazard. The alternative configurations from which the system controller can choose and perform are planned in advance by the planner in the form of plans (see definition 5), or of next operating point (NOP). Thus, we plan the system operation treating imminent failures, while assuring temporal and safety targets as introduced later in Section 5.4.

By reconfiguration, the system is transitioned from the COP to a NOP. Additionally, the current system state, and the predefined plans necessary for runtime reconfiguration constitute the plan model which is part of the system controller, as depicted in Figure 5.2. The latter emphasizes high-level view of the system controller workflow to perform the reconfiguration. The system controller, in regular operation, keeps observing the imminent hazards, reported by the diagnoser. Upon detection, the system controller selects the appropriate configuration to move the system into an appropriate NOP. As the BE zone is unprotected, the system controller first issues the reconfiguration of the BE zone, cooperating with the BE controller and setting a timeout on the response time of actions performed by the BE controller. That way, the reconfiguration of the SC zone is not issued, while unprotected zone is being reconfigured, ensuring isolation and sufficient independence between activities within BE and SC zones (as detailed in Sections 5.3.1 and 5.3.2), satisfying *Req*1. Moreover, the isolation between the reconfigurations

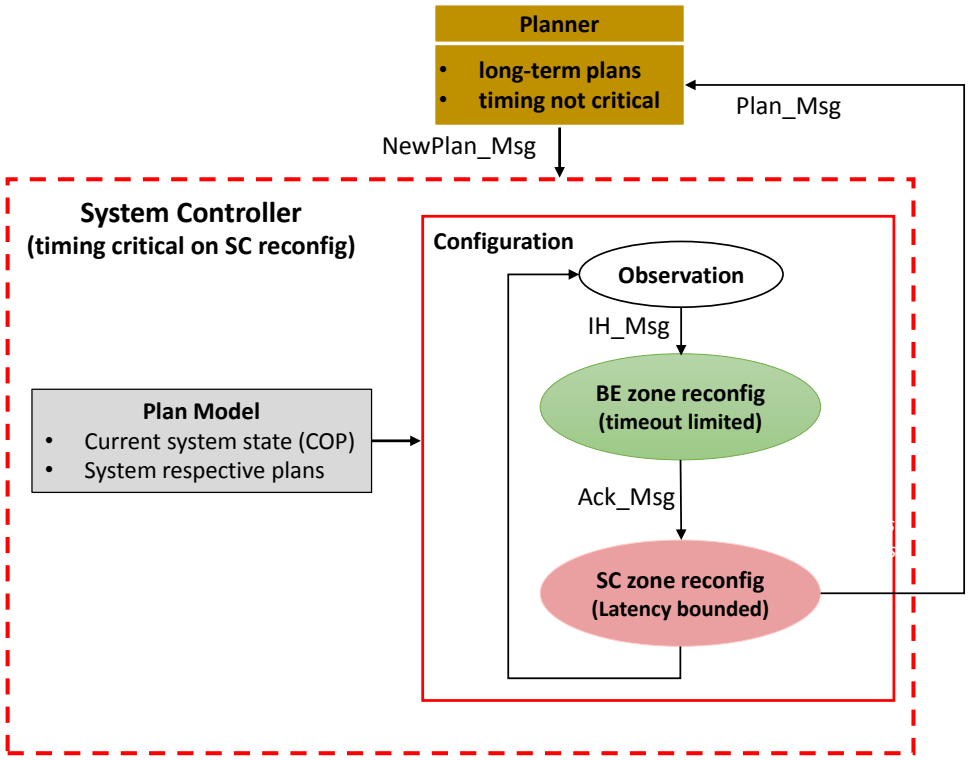


Fig. 5.2.: High-level view of the system controller functionality, cooperating with the planner and local controllers at runtime

of BE and SC zones provides minimum service disruption to timing-critical tasks as they are only subject to time required to configure the SC zone, increasing the feasibility of our approach. After the reconfiguration is preformed, the system controller signals the planner via the plan message *Plan_Msg* to plan and provide the corresponding plans for the new state. The planner in turn reports back the new plans via *NewPlan_Msg*. Table 5.1 presents an overview of different message types exchanged between the system controller and local controllers.

Any plan involves controllers' collaboration. Freeing BE resources under dynamic load distribution requires finishing BE internal migration under control of the BEC before the system controller can move the boundary. If the BEC is unable to provide such collaboration, plan execution is not possible. Note that non-collaboration requires the unlikely coincidence of two effects, IH and BE error, but it must be included in the safety concept as operation of the BE part is not guaranteed. In this case, there are two options, (a) continuing system operation just reporting the two effects or (b) discontinuation of BE service and continuation of system reconfiguration. The former choice depends on the application system and is beyond this work. Here, we assume (b) as the extended strategy.

Table 5.1.: The communication protocol messages: types and descriptions

Message Type	**Message**	**Description**
Control Messages	*SveState_Msg*	save the state *(SveState)* message, sent from the system controller to the CTC in order to save the state of the failing tile
	RstRsm_Msg	restore *(Rstore)* message, sent from the system controller to the CTC to restore the state on the identified BE tile and resume the execution of the SC container
	Ack_Msg	Acknowledge *(Ack)* messages, e.g., from BE controller/CTC to the system controller to indicate readiness
	Clear_Msg	clear message, sent from the system controller to the BE controller in order to clear the identified tile from the BE tasks
	Cfg_Msg	configuration *(Cfg)* message, sent from system controller to the local actuators in the identified BE tile to reconfigure the peripherals.
Diagnose Messages	*IH_Msg*	imminent hazard *(IH)* message sent from the diagnoser to the system controller
Planning Messages	*Plan_Msg*	sent from the system controller to the planner in order to start planning for the new system state
	NewPlan_Msg	sent from the planner to the system controller to indicating release of new plans

5.3.1. Best-Effort Zone Reconfiguration

The reconfiguration starts with the BE zone, where a healthy tile that is suitable for replacement must be identified, as depicted in Figure 5.3 via the identification process. Suitability includes timing and ability to adhere to safety requirements when configured properly. It requires pre-selection of suitable components and health check (test and diagnose) before release. As previously mentioned, the potential BE destinations are provided by predefined plans which are stored in the system controller database. The BE tile is not chosen to be in the solution space unless it is ensured the migration to it satisfies timing constraints (cf. Section 5.4). Based on that, the system controller chooses the planned BE destination, however, it must ensure the health state of the chosen one. This step is architecture dependent with similar requirements to health checking as in existing repair procedures for critical systems. As BE tile identification is not on the critical path or reconfiguration, it is not further investigated in this work.

Figures 5.3 and 5.4 depict the communication protocol between local and global controllers required to respectively perform cooperative runtime reconfiguration in BE and SC zones. Once a detector foresees a core failure on a SC tile, it sends an imminent hazard message *IH_Msg* to the system controller. The communication is performed by posting the message in the shared main memory region (dedicated to communication between the detective/preventative components in both BE and SC zones), and employing inter-processor interrupts. Figure 5.3 depicts that the system controller commands to the BE controller by sending the clear message *Clear_Msg* to prepare the BE tile for migration. Consequently, the BE controller stops the BE tasks execution on the identified BE tile, migrates them to other BE tiles, and finally sends an *Ack_Msg* to the system controller. Furthermore, when the BE tile is ready, the system controller gets the control back over the system. It reconfigures remotely the peripherals in the identified BE tile by sending a *Cfg_Msg* to the actuators. The latter disable required peripherals, turn the tile to an idle mode, and acknowledge the system controller via an *Ack_Msg*. These actions must be bounded and referred to as D_{CF}, as detailed in Section 5.4. After that, the system controller validates the progress of the BE reconfiguration through ensuring the delivery of *Ack_Msgs* (as detailed in Section 5.3.2). If the requested *Ack_Msg* has not been delivered from the BE controller after a timeout, reconfiguration is enforced (option (b), see above).

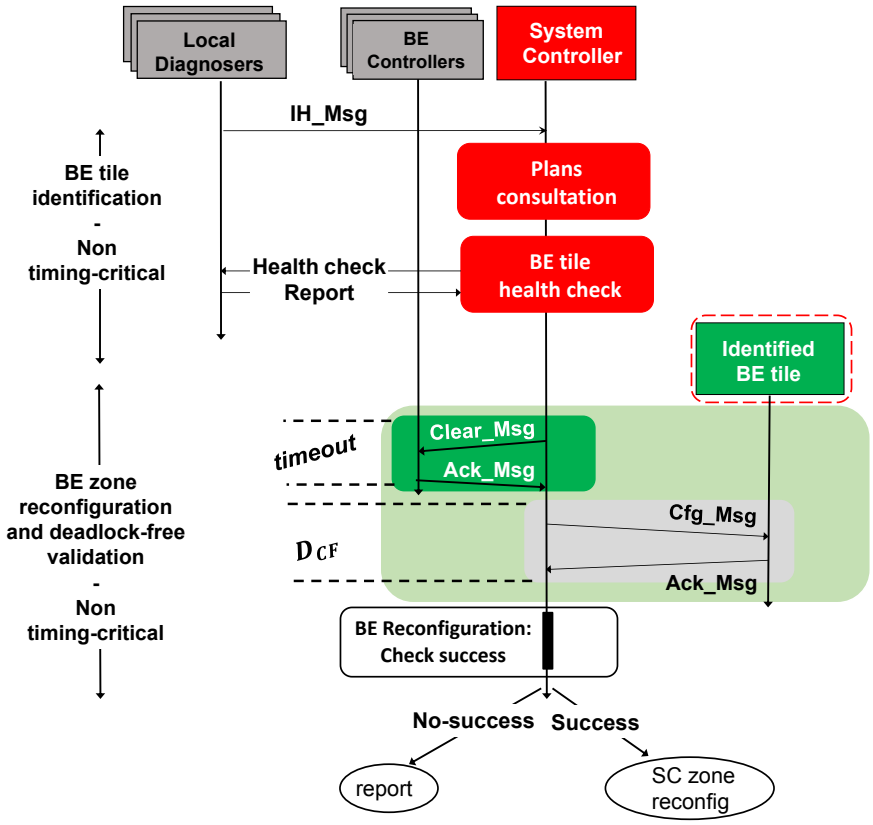


Fig. 5.3.: Deadlock-free reconfiguration of the BE zone employing the communication protocol

5.3.2. Safety-Critical Zone Reconfiguration

In general, when we stop the execution of the safety-critical tasks running on the failing tile, we distinguish two preemption schemes: task-level and job-level. In case of task-level preemption, the execution may be stopped only after completing the current job. This will unnecessarily increase the latency overhead of the overall reconfiguration as we need to wait for the job to be completed. Besides, following this approach, we need to account for all other preempted tasks to finish their jobs. Therefore, we preferred to develop an agile reconfiguration approach employing the job-level preemption, where the execution of a task could be stopped directly before completing the current job. As depicted in Figure 5.4, the SC zone reconfiguration is mainly composed of three stages, *containers remapping*, *state saving*, and *execution resuming*. Moreover, the system controller ensures a deadlock-free progress at the end of each reconfiguration step via the *check success* process, assuring *Req4* (cf. Section 5.1).

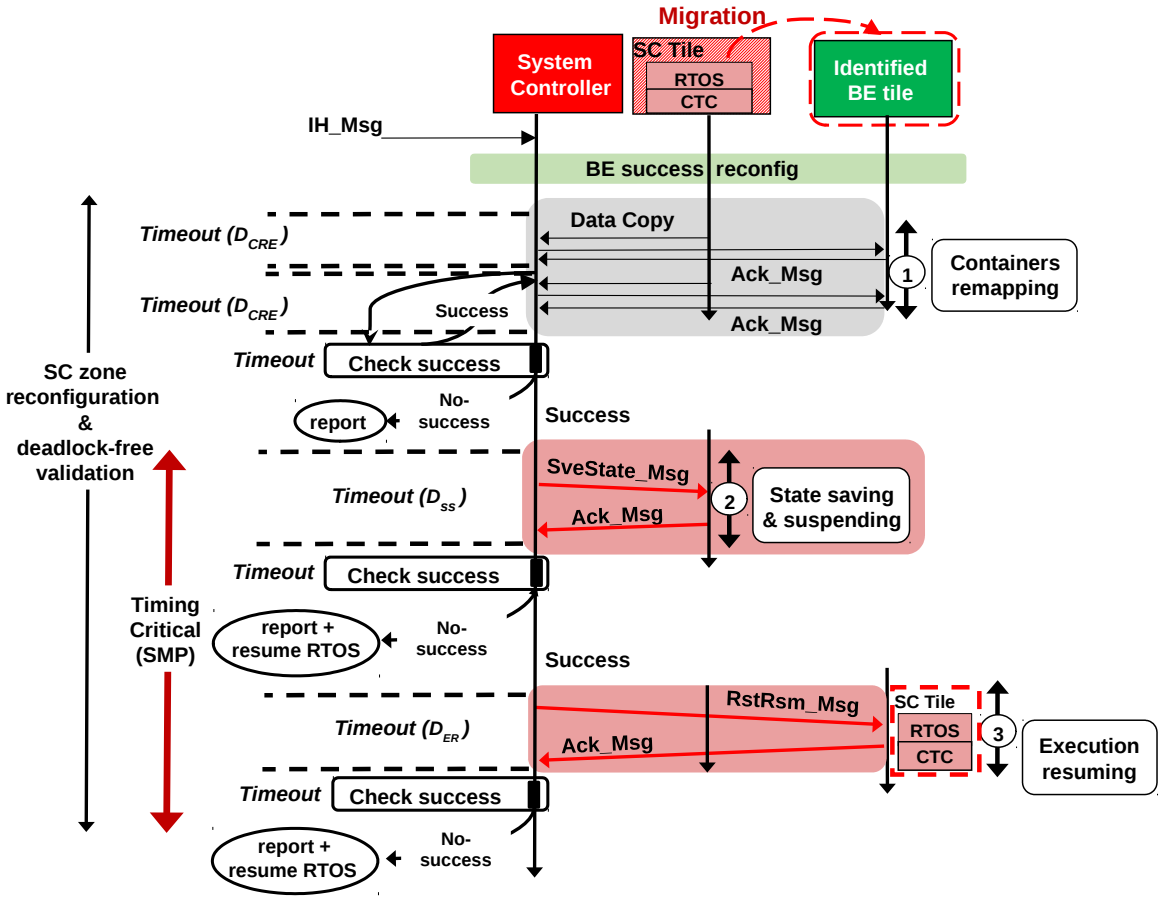


Fig. 5.4.: Deadlock-free reconfiguration of the SC zone employing the communication protocol

Containers remapping

At this stage, data including RTOS, application tasks, and interrupts related data are copied from the failing tile to the identified BE tile. As the failure has not occurred yet, the data is copied rather than being moved, preserving the continuous operation of the tile, whose failure is foreseen. The system controller migrates (copies) the SC container to the identified BE tile, where RTOS is resumed on the BE core at runtime. To this end, the system controller must adjust shared resources – the NoC – for adopting new system changes, which require a NoC with reconfigurable isolation between BE and SC zones. The system controller reprograms the NIs entries of both failing and identified tiles – performing migration (data copy). Thus, the system controller's functionality adheres to the highest relevant safety level to ensure a safe function of the proposed approach.

The NI acknowledges the system controller upon proper write via an *Ack_Msg*. After that, the system controller investigates the proper execution of these reconfiguration actions via requesting an *Ack_Msg* delivery at the end of each reconfiguration action. The global controller might not receive an *Ack_Msg* within a

maximum allowed time from a local controller/NI due to, e.g., NoC dropping messages. Thus, the reliable transport mechanism at transport layer is employed in our approach, where the system controller requests the *Ack_Msg* delivery, implementing the principle of the stop-and-wait (SNW) protocol [136]. Upon the respective *Ack_Msg* delivery, the system controller continues to the next reconfiguration step, ensuring deadlock-free progress and satisfying *Req4* at this reconfiguration stage. However, during the *check success*, if the *Ack_Msg* has not been delivered after a timeout, the system controller reports the issue, concludes the reconfiguration, and continues system operation (as the IH has not occurred yet). Such misbehavior is very unlikely and might refer to an error occurrence in the SC part beside the reported IH (co-exist of an error and IH) which is beyond the scope of this work. Note that for sake of simplicity, Figure 5.3 and 5.4 do not depict the SNW protocol that is incorporated in the *check success* process. Also, Figure 5.4 exemplifies two NI writes that respectively trigger *check success* twice. Recall that bounding the timeout for each reconfiguration step is detailed in Section 5.4.

Moreover, Figure 5.5 exemplifies the migration process through two tiles accessing the main memory via NoC. The NI supports memory management and interrupt-based inter-tile communication through an address translation table (ATT) and an interrupt translation table (IRQTT), respectively. The NIs control the inter-tile communication as tiles can only communicate with other tiles and resources if they are allowed by the system controller that maintains the configuration of the NIs. Moreover, the NI can also enforce spatial isolation by means of a memory protection unit. The ATT transparently translates the tile local addresses into a global address space, where local requests are translated to remote accesses and transmitted over the NoC. The ATT consists of entries and each entry contains a tile local address, a corresponding remote (Rmt) address, a route in the network (routing path), memory protection flags: read (R) and write (W), and the valid flag that indicates the entry validation. The required memory access protection layer is fulfilled via R and W flags, which control the nodes access rights to memory regions, satisfying *Req1* (cf. Section 5.1). The first table entry in the SC tile's NI points to the SC region in the main memory where the SC container is. The second entry points to the shared memory region between the SC and BE tiles. The ATT in the BE tile's NI points to the BE and shared regions in the memory in the same manner.

In the presence of a foreseen SC tile failure, the system controller migrates (copy) the SC container to the BE tile by reconfiguring the NIs to point the ATT of the identified BE tile to the SC memory region. It performs the process through two steps. First, it remotely reads the first entry of the ATT in the SC tile. Second, it updates the first entry of the ATT in the BE tile by remotely writing the previously

read entry. Consequently, the first entry of the ATT in the BE tile points to the SC region in the memory – indicating success migration.

Moreover, the system controller also reconfigures the IRQTTs of both BE and SC tiles in the same manner. It reconfigures its table as well. The IRQTT includes the same parameters of the ATT except that the memory addresses are replaced with IRQs local and remote numbers. The IRQTT basically translates a local tile interrupt number to a remote number in a destination tile. Due to migration, the interrupts sent from the system controller to the SC tile must be redirected to the identified BE tile as it now represents the new SC tile. Similarly, the interrupts entries in the SC tile's NI must be copied to the BE tile's NI. Note that after migration, the SC tile replaces the identified BE tile (as presented in the timeline in Figure 5.4), which implies that the tile is now hosting the SC container. That way, new BE and SC boundaries are established, resulting in NoC with reconfigurable isolation feature, satisfying *Req2* (cf. Section 5.1).

State saving

At this point, the system controller triggers a sensitive migration path (*SMP*) under which the sensitive configurations is performed, impacting the temporal guarantees of timing-critical tasks. The *SMP* must be also upper-bounded and referred to as MLO^{SMP} (maximum latency overhead), as detailed in Section 5.1. The system controller first commands to the CTC to save the current state of the failing tile by *SveState_Msg*. The CTC saves the state of the processor registers and the peripherals in the shared main memory. In addition, the CTC suspends the execution of the running RTOS and SC tasks on the failing tile, and signals readiness to the system controller via an *Ack_Msg*. Similar to previous stage, the system controller triggers the *check success* to assure the *Ack_Msg* delivery. In case of improper delivery of the *Ack_Msg* employing the SNW protocol and after a timeout, the system controller brings the system back to normal operation by resuming the suspended RTOS and reports the issue.

Execution resuming

Once the system controller has pointed the identified BE tile to the corresponding SC memory region, it commands the CTC on the identified tile via *RstRsm_Msg* in order to restore the tile state from the shared memory and resume it. The CTC retrieves the stored processor and peripherals states from the memory to the identified tile, loads the registers with the corresponding values, and eventually resumes the SC container on the identified BE tile. That way, the RTOS has

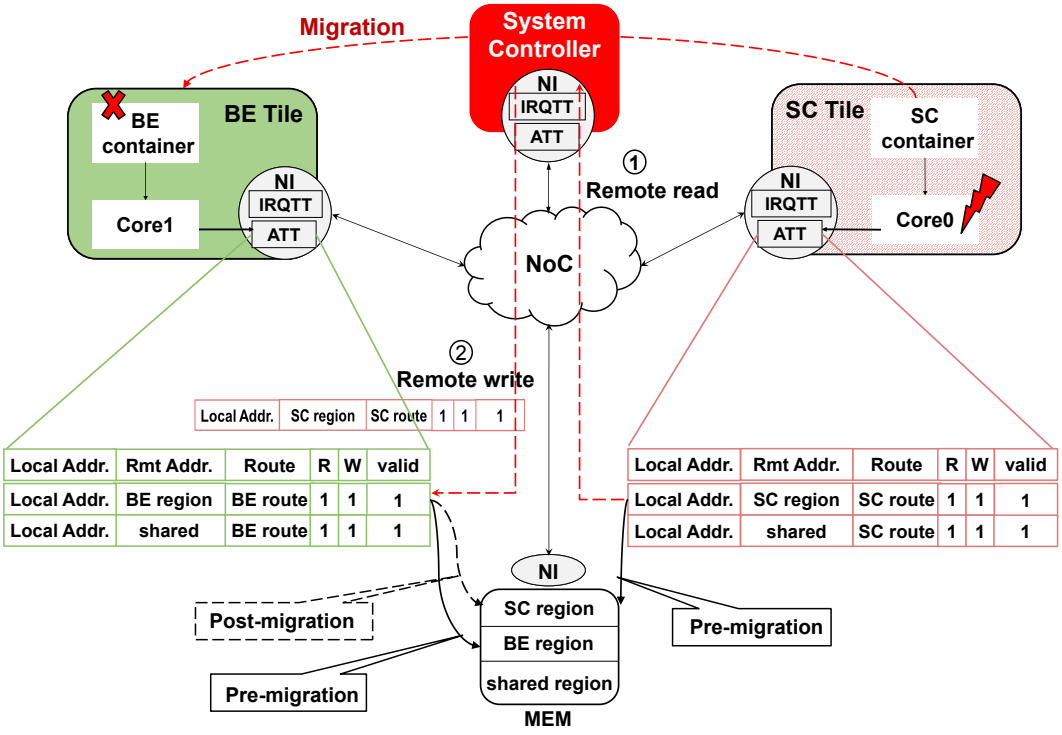


Fig. 5.5.: Example of the migration process (data copy) via reconfiguring the NIs' address translation tables

been resumed on the identified tile that was not running it in regular operation, preserving the required isolation between SC and BE zones (satisfying *Req1*). Finally, the system controller checks the success of these reconfiguration actions similarly to previous stages.

5.4. Temporal Analysis

To choose a BE tile destination for a potential migration, we need to assure temporal guarantees of the migrating SC tasks that will run on this BE tile in normal operation. This could be easily computed employing system-level timing analysis tools such as CPA [68, 155]. On the other hand, choosing the BE tile by the planner also requires bounding the latency overhead needed to migrate the SC container to this tile, ensuing all firm deadlines are met during migration. To this end, this section provides a formal analysis framework used by the planner to derive the worst-case latency overhead induced by migration to a potential BE tile destination. Next, the upper-bounds of the response times of timing-critical tasks, accounting for migration, are computed. After that, the permissible *deadline slacks* are extracted and only the BE destination to which the migration induces positive slacks is chosen, satisfying *Req3* (Section 5.1).

Definition 15. *Task's deadline slack $DSlack_i$ defines the time budget between the task's deadline D_i and its worst-case response time, including the latency overhead of our reconfiguration approach $R_i^{MLO^{SMP}}$, where MLO^{SMP} refers to the worst-case duration of SMP. It is computed as follows:*

$$DSlack_i = D_i - R_i^{MLO^{SMP}}, \tag{5.1}$$

where the constraint $DSlack_i > 0$ must be satisfied for each migrating task that is timing-critical.

In the following, we detail the computation of the MLO^{SMP}. We first introduce some terms used in the analysis. the term $M^{rreq}_{SyC \to M}$ denotes a remote read request from the system controller (SyC) to the memory (M). Similarly, $M^{wr}_{SyC \to NI(CTC)}$ denotes the remote write from SyC to the network interface on the tile of the CTC , while $M^{IRQ}_{SyC \to CTC}$ denotes an inter-processor interrupt from the core of SyC to the core of CTC . Before focusing on the 2 phases constituting *SMP*, let us first discuss the worst-case latencies of communication messages. We assume for the network communication (tile-to-tile and tile-to-memory) the slot-based transmission (SBT-NoC) mechanism[1], which relies on a dedicated bus-based medium to arbitrate between NoC access requests [123]. Note that our approach can be combined with other NoC architectures and analysis frameworks, such as compositional performance analysis [155], network calculus [26], integer linear programming based analysis [31], and real-time calculus [168]. However, in this work we decided to employ the SBT-NoC approach [123]. The motivation for our choice is that this is a recent work developed around realistic NoC assumptions and properties. Furthermore, we consider that migration-related actions always have higher priority than regular computation-based jobs of tasks. Hence, the worst-case latencies of individual messages are addressed as follows.

The worst-case latency of a message M_i can be obtained by solving Equation 5.2.

$$M_i = C_i + O_i + A_i, \tag{5.2}$$

where C_i presents the transmission of the message in isolation (i.e. without any contentions), O_i presents the worst-case out-of-slot arrival penalty when a message arrives just after the new slot has started, and A_i presents the arbitration delay which is equal to one slot, consistently with [123]. Eq. 5.2 presents an adapted version of Eq. 9 from [123] where the interference term is excluded. In our approach, all messages exchanged during *SMP* are transmitted with the highest

[1]For brevity, we do not provide a detailed description of the SBT-NoC mechanism. Instead, we only focus on its aspects that are crucial for presenting our approach.

priority, and *SMP* serialises its phases and actions. This implies that no two *SMP*-relevant messages will interfere with each other. Since the interference component is always equal to zero, it can be excluded from consideration and hence Eq. 5.2 presents the worst-case traversal time of message M_i. Now, let us focus on bounding the durations of the phases constituting *SMP*.

Component 1 – State saving

Let D_{SS} denote the worst-case duration of the state saving phase. It can be computed by solving Equation 5.3.

$$D_{SS} = M^{wr}_{SyC\rightarrow M} + M^{IRQ}_{SyC\rightarrow CTCF} + M^{rreq}_{CTCF\rightarrow M} + M^{rres}_{M\rightarrow CTCF} + C^{S\&P}_{CTCF} + M^{wr}_{CTCF\rightarrow M} + M^{IRQ}_{CTCF\rightarrow SyC} + M^{rreq}_{SyC\rightarrow M} + M^{rres}_{M\rightarrow SyC} \quad (5.3)$$

As shown in Equation 5.3, this phase consists of bounding 3 sets of actions. In the first line, SyC sends a command to CTC in the failing tile (*CTCF*) to save its state. This is performed via the memory write by SyC, then an inter-processor interrupt to *CTCF* and finally a read from the memory by *CTCF*. In the second line, *CTCF* stops the execution of tasks on its own core, and creates a message where an entire execution context is stored. Then, *CTCF* writes this context to memory. The term $C^{S\&P}_{CTCF}$ denotes the delay of stopping the execution, creating the message and populating its payload with the execution context information. The context corresponds to processor general and special-purpose registers, and peripheral states, e.g., timer and generic interrupt controller. Finally, the third line derives the maximum time it takes for sending the acknowledgement message from *CTCF* to SyC. This is executed via an inter-processor interrupt and a read from memory performed by SyC. In case the acknowledgement from *CTCF* to SyC is successfully delivered, the state saving process concludes. However, recall from the previous section that in some scenarios an acknowledgement message may not be successfully delivered. In such cases, an additional sub-phase called *check success* is executed. Its worst-case latency that sets its timeout, denoted by $D^{SyC\rightarrow CTCF}_{SUCC\&RSM}$, can be computed by solving Equations 5.4 and 5.5.

$$D^{SyC\rightarrow CTCF}_{SUCC} = M^{wr}_{SyC\rightarrow M} + M^{IRQ}_{SyC\rightarrow CTCF} + M^{rreq}_{CTCF\rightarrow M} + M^{rres}_{M\rightarrow CTCF} + C^{V\&A}_{CTCF} + M^{wr}_{CTCF\rightarrow M}$$

$$+ M^{IRQ}_{CTCF \to SyC} + M^{rreq}_{SyC \to M} + M^{rres}_{M \to SyC} \tag{5.4}$$

$$D^{SyC \to CTCF}_{SUCC\&RSM} = D^{SyC \to CTCF}_{SUCC} + M^{ENB}_{SyC \to FT} \tag{5.5}$$

This sub-phase consists of 3 sets of actions (Equation 5.4). The first line bounds the time needed by SyC to request the *Ack_Msg* from CTC*F* via shared memory communication and interrupts. The second line derives the time required by CTC*F* to re-validate the correctness of the performed state saving operation and prepare the *Ack_Msg*. The worst-case latency of this set of on-core operations is denoted by $C^{V\&A}_{CTCF}$. Finally, the third line corresponds to *CTCF* that sends an *Ack_Msg* to SyC, and SyC retrieves it from the memory. In addition, in the worst-case the *Ack_Msg* might not be delivered. Consequently, Equation 5.5 derives the maximum time required for check success $D^{SyC \to CTCF}_{SUCC}$, and as well bounds the maximum time required by the SyC to send a direct interrupt message to the failing tile (FT) to resume the suspended RTOS and preserve the system operation $M^{ENB}_{SyC \to FT}$.

Component 2 – Execution resuming

Let D_{ER} denote the worst-case duration of the execution resuming phase. It can be computed by solving Equation 5.6.

$$D_{ER} =$$

$$M^{wr}_{SyC \to M} + M^{IRQ}_{SyC \to CTCI} + M^{rreq}_{CTCI \to M} + M^{rres}_{M \to CTCI} +$$
$$C^{U\&L}_{CTCI} + M^{wr}_{CTCI \to M} + M^{IRQ}_{CTCI \to SyC} + M^{rreq}_{SyC \to M} + \mathscr{M}^{rres}_{M \to SyC} \tag{5.6}$$

As shown in Equation 5.6, this phase consists of 3 sets of actions. In the first line, SyC sends a command to CTC in the identified best-effort tile (*CTCI*) to retrieve the state (execution context) from the memory. In the second line, CTC*I* reads the state message from memory, unpacks the execution context, prepares the relevant resources (e.g., loads the values in registers), and starts them. This is denoted with the term $C^{U\&L}_{CTCI}$. Finally, CTC*I* sends an *Ack_Msg* to SyC. Similar to the previous stages, if *SyC* does not receive the *Ack_Msg*, it initiates the *check success* sub-phase. Its worst-case latency is denoted by $D^{SyC \to CTCI}_{SUCC\&RSM}$ (see Equation 5.5).

This marks the completion of *SMP*. Now the worst-case duration of *SMP*, referred to as MLO^{SMP}, can be computed by summing up the aforementioned terms (Equation 5.7).

$$MLO^{SMP} = D_{SS} + D^{SyC \to CTCF}_{SUCC\&RSM} + D_{ER} + D^{SyC \to CTCI}_{SUCC\&RSM} \tag{5.7}$$

Note that during an entire *SMP* time interval, the affected/migrating tasks are not able to execute. When reasoning about the schedulability of these tasks, the migration time needs to be taken into account. In the schedulability analysis for on-core execution, we model this overhead with the highest priority job of execution time MLO^{SMP} that appears only once. We assume that on each tile there exists a fully preemptive uniprocessor with sporadic tasks with fixed priorities and constrained deadlines ($\forall \tau_i : D_i \leq T_i$). Hence the worst-case response time of task τ_i residing on a tile where the imminent failure was signalled can be computed by solving Equation 5.8.

$$R_i^{MLO^{SMP}} = C_i + \sum_{\forall \tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^{MLO^{SMP}}}{T_j} \right\rceil \cdot C_j + MLO^{SMP} \tag{5.8}$$

In Eq. 5.8, C_i refers to the execution time of a task in isolation, and term $hp(\tau_i)$ denotes a set of higher priority tasks sharing the core with the analysed task τ_i. Note that Eq. 5.8 is a slightly revised version of the well-known equation for the computation of response times of tasks on a fully preemptive uniprocessor [27, 30]. Eventually, to assure temporal guarantees of timing-critical tasks, the constraint $DSlack_i > 0$ must be satisfied for each critical task. Note that the known relative deadline for each critical task is an essential requirement in hard real-time systems [106], otherwise, meeting deadlines cannot be formally guaranteed.

5.4.1. Timing Impact on Non-Migrating Tasks

In the previous section we have computed the worst-case duration of the sensitive migration path (*SMP*) and also provided the schedulability analysis for tasks residing on the SC tile that is affected by the imminent hazard and hence is subject to migration. However, the migration can also affect tasks existing on other tiles (i.e., those not involved in the migration process). That is attributed to the fact that reconfiguration-related messages (belong to the sensitive migration path) imposed on NoC have higher priority than regular computation-based jobs of tasks and thus, in the worst-case, the latter are delayed by the former (especially due to the contention at memory) and their timing properties are affected. Hence, in this section we provide the schedulability analysis for such tasks.

To this end, let us first analyse the individual effects that migration-related operations might have on non-migrating tasks. When it comes to the execution of migration-related on-tile operations, they do not have any influence on tasks located on other tiles, simply because those operations are executed locally, on

the tile. However, the migration-related network traffic can interfere with tile-to-tile or tile-to-memory communication of other tasks, and eventually lead to longer execution of their on-tile operations. Hence, when reasoning about the schedulability of tasks, we need to take the potential interference from migration-related communication into account.

Let us calculate the cumulative latency of all communication messages induced by the phases constituting *SMP* (the *State saving* and the *Execution resuming* in Figure 5.4), termed *MRI* (maximum reconfiguration interference) as is in Equation 5.9.

$$MRI = MLO^{SMP} - C^{S\&P}_{CTCF} - C^{V\&A}_{CTCF} - C^{U\&L}_{CTCI} - C^{V\&A}_{CTCI}. \tag{5.9}$$

The term *MRI* is computed by subtracting the durations of all migration-related on-tile operations ($C^{S\&P}_{CTCF}$ *and* $C^{V\&A}_{CTCF}$ in the failing tile, $C^{U\&L}_{CTCI}$ *and* $C^{V\&A}_{CTCI}$ in the identified best-effort tile) from the duration of the entire latency overhead of the sensitive migration path MLO^{SMP}, computed earlier in Equation5.7.

Since all migration-related messages are of the highest priority, we conservatively assume that all tasks located on non-migrating tiles in the worst-case suffer the full migration-related interference. Hence, the worst-case response-time of any task τ_k, which is not located on the failing tile, can be computed by solving Equation 5.10.

$$R^{MRI}_k = C_k + \sum_{\forall \tau_l \in hp(\tau_k)} \left\lceil \frac{R^{MRI}_k}{T_l} \right\rceil \cdot C_l + MRI. \tag{5.10}$$

Note that, similar to Equation 5.8, in Equation 5.10 term C_k denotes the execution time of the task τ_k in isolation, while term $hp(\tau_k)$ denotes a set of higher priority tasks sharing the core with the analysed task τ_k.

The deadline slack $DSlack_k$ for the non-migrating tasks is computed as follows:

$$DSlack_k = D_k - R^{MRI}_k. \tag{5.11}$$

We also remind that the constraint $DSlack_k > 0$ must be satisfied for each non-migrating task that is timing-critical.

5.4.2. Timing Impact on the Deadlock-Free Mechanism

In the previous discussion, the focus was on bounding the *SMP*, and assuring temporal guarantees of timing-critical tasks. This section upper bounds the additional reconfiguration actions required by safety-critical and best-effort zones to

be employed by the system controller during the validation of deadlock-free configuration. Hence, the remaining reconfiguration-related parts like the container remapping (cf. Figures 5.4) and also best-effort reconfiguration (cf. Figure 5.3) are upper bounded. Let us first focus on bounding the latency overhead of the container remapping.

Containers remapping

Let D_{CR} denote the worst-case duration of the containers remapping phase. It can be computed by solving Equations 5.12 and 5.13.

$$D_{CRE} = M^{rreq}_{SyC \to NIF} + M^{rres}_{NIF \to SyC} + M^{wr}_{SyC \to NII} + M^{wr}_{NII \to M} + M^{IRQ}_{NII \to SyC} + M^{rreq}_{SyC \to M} + M^{rres}_{M \to SyC}. \tag{5.12}$$

$$D_{CR} = n \cdot D_{CRE} + C^{UP}_{SyC}. \tag{5.13}$$

As shown in Equation 5.13, this phase consists of n repetitions of D_{CRE} (cf. Equation 5.12) that bounds reading a NI's entry of the failing tile (NIF), writing it in the network interface of the identified tile (NII), and an *Ack_Msg* from *NII*, where n stands for the worst-case number of IRQTT and ATT entries that need to be copied. The second component C^{UP}_{SyC} represents an upper-bound of on-core actions performed by the SyC to update its own $IRQTT$ entries to now point to the identified (new SC) tile. As in the previous phase, any missed *Ack_Msg* trigger the *check success* sub-phase. It is very similar to Equation 5.4, with the only distinction that participants are different (in this case SyC and *NII*), and thus it is denoted by $D^{SyC \to NII}_{SUCC}$. Note that the containers remapping process might include at most n check success sub-phases, one for each of IRQTT and ATT entries.

Now we focus on the process of configuring the identified best-effort tile remotely in order to be prepared for migration (the gray part in Figure 5.3). Let D_{CF} denote the worst-case duration of the said configuration process. It can be computed by solving Equation 5.14.

$$D_{CF} = M^{wr}_{SyC \to M} + M^{IRQ}_{SyC \to BE} + M^{rreq}_{BE \to M} + M^{rres}_{M \to BE} + C^{CF}_{BE} + M^{wr}_{Act \to M} + M^{IRQ}_{Act \to SyC} + M^{rreq}_{SyC \to M} + M^{rres}_{M \to SyC}, \tag{5.14}$$

where the first line bounds the time required by SyC to send a Cfg_Msg to best-effort tile. C^{CF}_{BE} denotes the time required by the actuators (Act) in the identified BE tile to disable the peripherals by writing the disable data to their registers. Finally,

the third line bounds the time it takes to send an *Ack_Msg* from the actuators to SyC. Similar to the components of *SMP*, there could also be a *check success* phase where the system controller requests an *Ack_Msg* from the best-effort. The worst-case duration of this phase is denoted by $D_{SUCC}^{SyC \rightarrow BE}$ and it can be computed as in Equation 5.4. Thus, the system controller employs the presented timing factors to make a decision whether a validation of deadlock-free reconfiguration must be triggered, satisfying *Req4*.

5.5. Experimental Evaluation

5.5.1. Experimental Setup

For evaluation, we implemented the proposed approach using Gem5 system-level simulator [21]. In addition, the introduced analysis framework (Section 5.4) has been implemented in Python to derive the worst-case latency overhead of application tasks, accounting for the reconfiguration timing overhead. From this, the feasibility of our approach is derived when positive slacks are extracted (Equations 5.1 and 5.11). The hardware platform consists of tiles where each tile is associated with a single container. Inter-processor interrupts and DDR shared memory are employed as inter-tile communication, compatible with message passing interface (MPI) communication [134]. Furthermore, the NoC implements a deterministic XY source routing and wormhole switching. Therefore, packets are, prior to injection, divided into small elements called FLITs, and FLITs are injected in the NoC in rapid succession, where they travel towards the destination in a pipeline manner. The input buffer capacities can be of arbitrary size. The arbitration of NoC access requests is performed through a dedicated bus-based medium, consistently with [123]. Note that more details about the simulation setup are summarized in Table 5.2.

Moreover, our platform model does not consider local memories that similarly extend the presented analysis to bound the copy process of cache contents. We only introduce the fundamental basis that can easily be extended to incorporate caches in the future research. However, we give an overview about caches timing overheads. The cache contents in the tile, that is expected to fail, must be also copied by writing it back to the shared memory, and then retrieving it to the identified BE tile. The copy process will extend the container remapping phase in the SC zone reconfiguration (cf. Figure 5.4). Consequently, the analysis will be extended to bound the copy process of cache contents so that the system controller can employ this additional timing factor to check the success of the caches

Table 5.2.: Simulation setup

Processor	ARM Cortex-A7
RTOS	μCOS-II [104, 7]
Scheduling policy	Static-Priority Preemptive (SPP)
Network	2D Mesh - 4×4 NoC
Routing algorithm	XY source routing
Switching technique	wormhole switching
Link bandwidth	35bits/cycle
Flit size	140 bits
Router pipeline, ports	4-stage, 5-port

reconfiguration. Moreover, as the copy process is performed while the failing tile is still in operation, some data in the data cache might be changed. Consequently, only the timing overhead required to update these data in the identified BE nd tile is considered in the *SMP* rather than the higher timing overhead needed to copy all contents of instruction and data caches.

5.5.2. Simulation-Based Latency Overhead

We perform rigorous experiments to demonstrate the latency overhead of our reconfiguration approach employing an application example from automotive domain. However, the introduced approach is not limited to certain usecases nor architectural parameters. The applicability of the approach in cyber-physical systems is far wider because mixed-criticality systems on many-core platforms are used in all major areas with dependability requirements from aerospace to medical applications, industry 4.0 or building automation. Thus, irrespective to the application area, the feasibility of our approach (i.e., satisfying the introduced requirements in Section 5.1) must be ensured as presented earlier through providing planned robust reconfiguration and temporal guarantees.

The employed usecase corresponds to the control of a vehicle with assistance functions [80]. The vehicle is designed to recognize an unknown space by populating a database of obstacles. Cameras' data are analysed through background estimation and feature extraction (BFE) tasks, and fused through feature data fusion (FDF) tasks. The obstacles, obtained by stereo photogrammetry (STPH), position sensors (POSI), and ultrasonic sensors (USOS), are eventually managed by the obstacle database manager (OBMG). Figure 5.6 depicts the tasks graph (up) and the corresponding mapping (bottom) of the introduced usecase. Moreover, Task

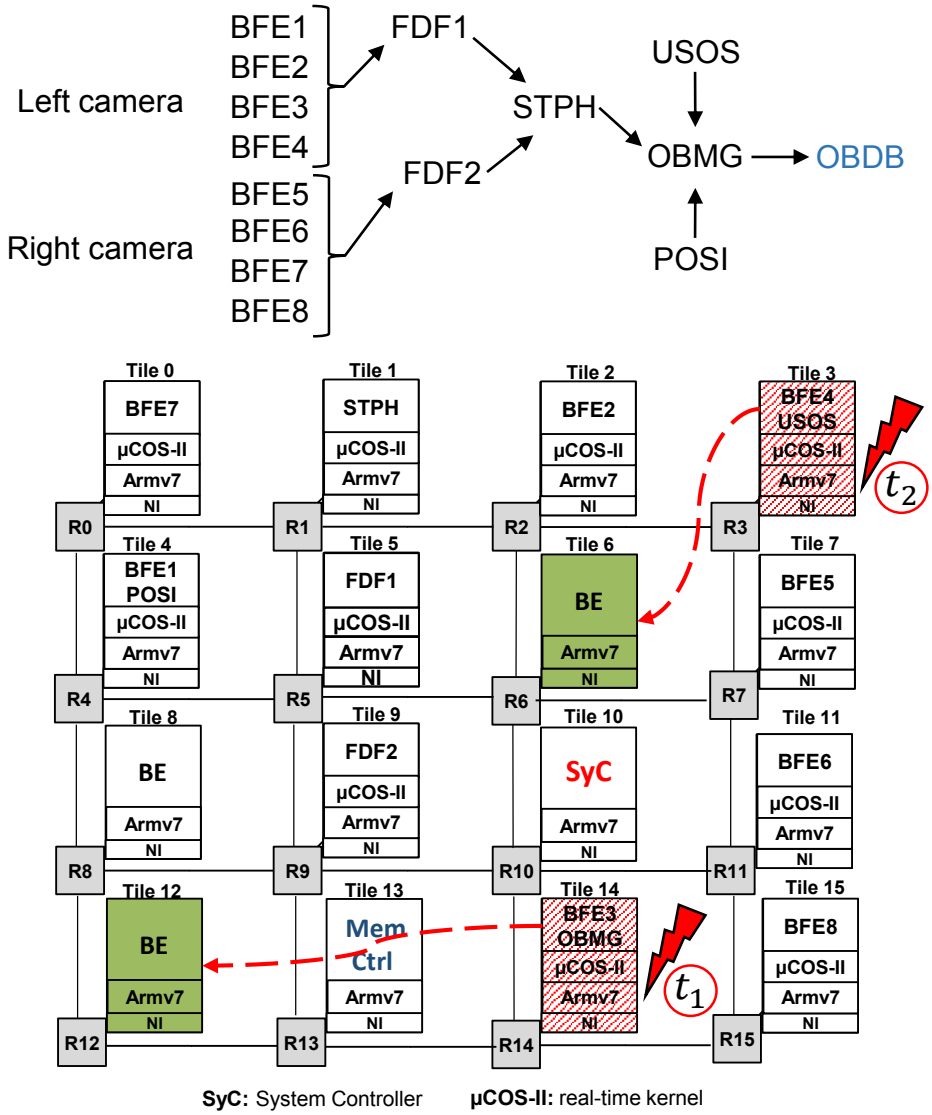


Fig. 5.6.: Graph (up) and mapping (down) of extracted tasks from the vehicle usecase [80]

periods vary between 40 ms to 1 s. The functionality selected for experiments is composed of 14 communicating tasks where all computations and transmissions are performed periodically. The usecase SC tasks can be mapped to a varying number of SC containers, here, they are mapped to 11 SC containers. Note that the communication between tasks in the tasks graph refers to data dependencies between tasks and not activation dependencies, and thus all tasks are triggered independently from each other.

We emulate the case of two tiles 14 and 3 are going to fail sequentially (at different points in time: $t_1 and\ t_2$), and thus the system controller reconfigures the system by migrating the SC containers on these tiles respectively to the identified BE tiles 12 and 6, according to provided plans. Noteworthy, in case of multiple imminent hazards are reported simultaneously, they will be served sequentially,

Table 5.3.: Comparison of reconfiguration requirements

	Ours	**[127]**	**[42]**	**[59]**	**[139]**
Req1: isolation	✓	×	×	×	×
Req2: protec. communication	✓	×	×	×	×
Req3: temp. guarantees	✓ (51.17%)	✓ / × (105%)	✓	× (240.3%)	×
Req4: robustness	✓	×	×	×	×

as the hazard is imminent and only will occur in the future. The reasoning beyond the aforementioned chosen failing tiles is related to the distance between the failing tiles, system controller and the memory. That is, CTC on Tile14 is closer to the memory and the system controller than Tile3 which requires longer time saving/retrieving data from/to the memory via NoC. That way we illustrate the reconfiguration temporal overhead with diverse failing tiles that are placed in various locations, resulting in diverse reconfiguration latency overheads. We employ the aforementioned formal analysis (cf. Section 5.4) on the selected usecase in order to derive the safe applicability of our approach under the presented two failing tiles scenarios. The worst-case response times of hard real-time tasks accounting for our approach are computed, validated against the tasks' deadlines (positive slacks), and thus the application of our approach is permitted.

Figure 5.7 plots the worst-case latency of the sensitive migration path (MLO^{SMP}) including the deadlock-free validation (Dfree) for proactively handling Tile14 failing. Also, the worst-case response times of the application tasks accounting for the *SMP* overhead ($R_i^{reconfig}$) are plotted. $R_i^{reconfig}$ corresponds to either $R_i^{MLO^{SMP}}$ (Equation 5.8) for the migrating tasks (BFE3 and OBMG), or R_k^{MRI} (Equation 5.10) for the other non-migrating tasks. In addition, the respective tasks' deadlines (D_i), and the extracted deadline slacks ($DSlack_i$ from Definition 15 and Equations 5.1 and 5.11) are also plotted. As shown, the applicability of our approach is extremely given due to high achieved deadline slacks. Similarly, the worst-case analysis-based results have been conducted to account for proactively handling Tile3 failing. For this, the worst-case response time including the maximum reconfiguration latency dedicates only 51.17% of the respective deadline, indicating as well the feasible application of the approach.

To position our approach, a comparison of reconfiguration challenges/requirements in mixed-criticality with other methods has been conducted. Table 5.3

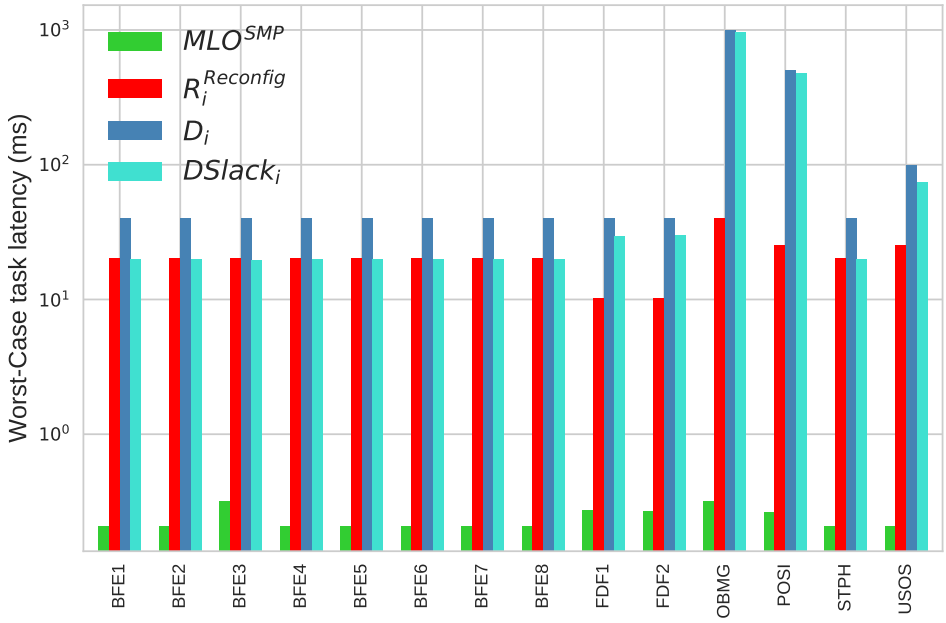
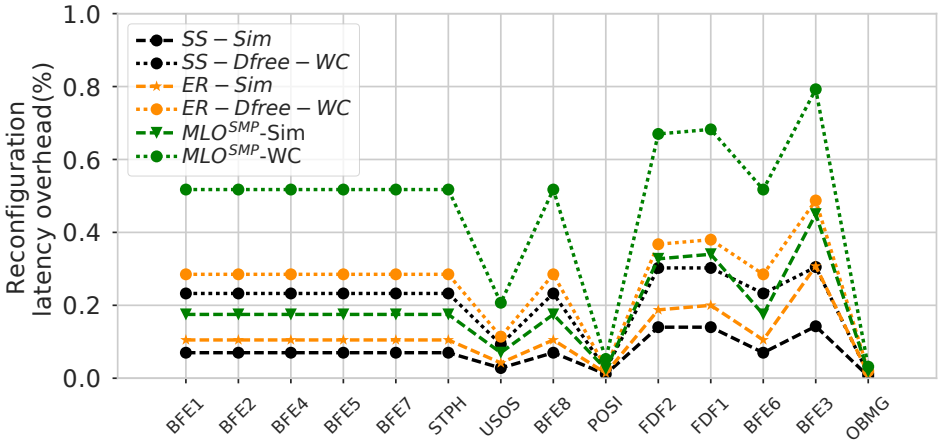


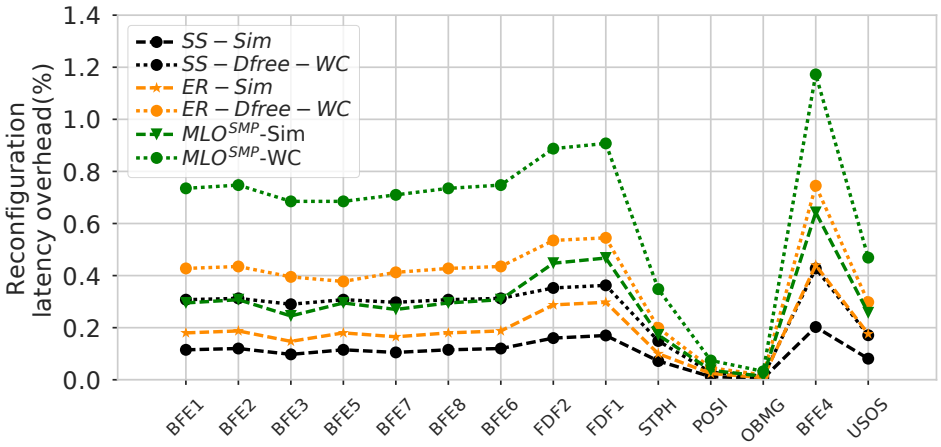
Fig. 5.7.: Analysis-based worst-case tasks latencies of the automotive usecase, considering an imminent hazard affecting Tile14

illustrates that other research studies do not feature isolation, protected communication channel, and robustness during reconfiguration, however, some of them support temporal guarantees. In our usecase, the worst-case response time including the maximum reconfiguration latency does not exceed the respective deadline in both Tile14 (50.79%) and Tile3 (51.17%) failing scenarios, outperforming [127] and [59]. In [127], although deadlines are met in some scenarios, however, the authors report that with an automotive usecase the maximum worst-case response time including the reconfiguration latency overhead (105%) still exceeds the application's deadline. Moreover, [59] does not report the analysis-based results, but states that the smallest deadline overrun of a critical application employing their adaptation strategy is 240.3% of the respective deadline, which is not allowed in hard real-time systems.

We now demonstrate the distribution of the worst-case reconfiguration latency overhead that impacts the timing of real-time tasks via the introduced two phases, *state saving* and *execution resuming*. Also, we conduct simulations in Gem5 to derive the simulation-based reconfigurations overheads. Figures 5.8a and 5.8b respectively plot the relative simulation (Sim) and analysis (WC) based reconfiguration latency overhead to the tasks' deadlines. The Figures illustrate the reconfiguration latency overhead incurred by *state saving* (*SS*), *execution resuming* (*ER*), and the sum of their latencies (MLO^{SMP}) required to proactively handle Tile14 (BFE3 and OBMG) and Tile3 (BFE4 and USOS) foreseen failures. Moreover, to give a better illustration of our approach, the impact of the reconfiguration



(a) Tile14 (RFE3, OBMG) foreseen failure



(b) Tile3 (BFE4, USOS) foreseen failure

Fig. 5.8.: Simulation and analysis based distribution of reconfiguration latency normalised to tasks' deadlines upon Tile14 (a) and Tile3 (b) foreseen failures

overhead has been plotted for all other tasks (non-migrating ones). Note that the time the system controller requires to choose an alternative configuration (i.e., container remapping) from the available plans is pretty swift and abstracted from the Figures. Other tasks also suffer from the reconfiguration-related messages imposed on NoC from the *SS* and *ER* phases as these messages have the highest priorities.

The varying impact of the reconfiguration latency overhead between tasks highly depends on the number of contentions at the memory controller and as well shared NoC paths between the tasks and the system controller. As depicted in Figure 5.8a, the maximum worst-case latency overhead induced by the reconfiguration due to Tile14 failing constitutes only 0.8% of the deadline (BFE3). OBMG task also suffers from the same reconfiguration latency, however, the relative overhead to its deadline (1 s) is very low. Likewise, Figure 5.8b depicts that the maximum worst-case reconfiguration latency constitutes 1.2% of the relative deadline (BFE4). In

the same manner, tasks BFE4 and USOS in Tile3 endure the same reconfiguration latency overhead, however, the relative overhead to the USOS's deadline (0.1 s) is also low. Thus, the proposed approach enables deterministic and robust runtime adaptations to system changes through reconfigurations with affordable latency overheads.

5.5.3. Implementation and Resource Overhead

The introduced controllers, the system controller, the critical tile controllers, and the best-effort controllers are implemented in software. One core is determined to execute the system controller as it must be active all the time and is timing-critical. The critical tile controller extends the real-time operating system (μCOS-II). That is, we have modified the μCOS-II to incorporate the critical tile controller so that it schedules and controls the safety-critical tasks accounting for the system controller indications. In the same manner, the best-effort controller extends the general purpose operating system to manage the best-effort workload execution under the supervision of the system controller.

The evaluation of the area and power overhead of the basic NI and NI with employing our approach (NIs_Reconfig) is carried out using ASIC design flow [89]. The TSMC 28*nm* process technology with high and standard threshold voltage in worst case corner ($WC, 0.72V, 125°C$) is applied. The results are obtained using the VHDL implementation of the IDAMC platform for ASICs [89], where NI components are synthesized, placed and routed, and then the parasitic extraction was performed. The primary hardware components of the NIs_Reconfig that induce additional overhead compared with the basic NI are the address translation table and the interrupt translation table. These tables incorporate higher number of entries than the normal case (Basic NI) to adopt new address and interrupt translation entries to perform container migration between the failing safety-critical tile and the identified best-effort tile. This increase of the tables' entries results in relative increase in the area and power overheads.

As depicted in Table 5.4, the size of all network interfaces (NIs) increases by 0.51% incurred by moving boundaries between best-effort and safety-critical subsystems. Moreover, the breakdown of the respective power figures is depicted in the table. The total power consumption of all NIs increases by only 1.08% employing our reconfiguration approach, which is quite tolerable.

Table 5.4.: Area and power overhead of system online reconfiguration

	NIs	**NIs_Reconfig**	**Relative Increase**
Area (um^2)	4944992	4970162	0.51%
Static Power (mW)	1.56	1.5857	1.62%
Dynamic Power (mW)	73.84	74.6387	1.07%
Total Power (mW)	75.4	76.225	1.08%

5.6. Summary

Embedded system platforms have grown considerably in complexity, and they continue to grow. They run large and evolving applications on heterogeneous multi- or many-core processing platforms. Such systems are required to provide dependable operation for the user by dealing with system degradation. This chapter presented an online resource management that supports treatment of system degradation (e.g., imminent core failure). Dynamic management of the system enables the system to plan and handle changes to the environment, the workload, and to the system itself at runtime. It also provides continuous operation, while ensuring guaranteed service even under strict safety and availability requirements. To achieve this, we introduced the set of controllers that operate in synergy in the system under the guidance of the global controller (the system controller) and the self-aware planning component (the planner).

The new concept of imminent hazards on a chip has been introduced, whose detection allows to proactively eliminate an unacceptable risk of future hazards. The hierarchical control approach providing online adaptations to prospective core failures in NoC-based mixed-criticality systems has been proposed. The proactive handling takes place upon the detection of an imminent hazard/failure. The alternative configurations from which the system controller can choose and deploy are planned in advance by the planner in the form of plans. The approach supports the system with adaptive and proactive deterministic hazard prevention protocols. It enforces predictable reconfigurations through container migration via enabling movable boundaries between safety-critical and best-effort subsystems. To enable that in NoC-based systems, NoC with reconfigurable isolation between traffics with different criticality levels has been supported.

In addition, another challenge of runtime reconfiguration is temporal guarantees of real-time tasks. To this end, we derived the worst-case latency overhead of the reconfiguration approach employing the introduced formal analysis framework (cf. Section 5.4). Doing so, the execution of runtime reconfigurations is permitted with positive application's slacks. Also, as the system controller interferes with a running system, it might bring the system to a deadlock/failure due to erratic behavior during reconfiguration. To remedy this, the reconfiguration approach is supported with *check success* sub-phases that validate the proper execution of the reconfiguration actions. Thus, to achieve a safe move between the boundaries over the state-of-the-art, the approach satisfied essential requirements during reconfiguration including isolation, protected accesses to shared resources, temporal guarantees, and deadlock-free reconfiguration.

The feasibility of the approach is experimentally demonstrated employing simulation and formal analysis tools to respectively evaluate average and worst-case scenarios through an automotive usecase. The results indicate that the runtime adaptation to changes to the system implies solely low allocations overhead and affordable latency. Although the hierarchical control approach has been elaborated to tackle imminent hazards, it can also handle the introduced two cases, change and error, through optimizing the presented timing scale to the respective case.

Chapter 6: Conclusions

To cope with the ever-increasing complexity of interconnected functions, and to reduce the cost and energy consumption of a system, multi- and many-core platforms used in mixed-criticality applications are emerged to efficiently integrate multiple processing elements on the same chip. These architectures are implemented with billions of transistors on a single die, or even on multiple chiplets integrated at the package level. Networks-on-chip (NoCs), as scalable and modular interconnects, are employed as a promising solution for MPSoCs. They constitute the core part of such systems and host traffic with different criticality levels.

In such complex chips, the associated high energy consumption is challenging and might lead to reliability issues. The energy issue can no longer be considered mainly due to the cores, as the energy contribution of a NoC to the overall energy budget is significant. Especially for safety-critical hard real-time NoCs, such as automotive and avionic domains, this is of high importance, as high energy consumption and the resulting thermal runaway can challenge the whole system guarantees [81, 78]. The efficient and safety-compliant energy management of NoCs is crucial for the success of MPSoCs in the market of safety-critical hard real-time systems. NoC energy control must account for dynamic system behavior in a dynamic environment. Besides, NoC reconfiguration and deployment of new energy setups must assure temporal guarantees of critical functions (i.e., meeting their deadlines).

Moreover, the control of such multiprocessing architectures becomes more complex under abnormal situations (e.g., system degradation). These systems are exposed to changes due to, e.g., aging and dynamic environmental operations. Thus, future many-core systems must not only manage the resource usage at runtime, but also cope with various changes like high energy consumption and system degradation. For mixed-criticality systems, dynamic resource management and degradation treatment must account for temporal constraints and strict isolation requirements between critical and non-critical functions [81, 78, 49].

Solving the two crucial challenges concerning energy management of hard real-time NoCs and degradation treatment in mixed-criticality systems was the major

contribution of the presented work. This thesis proposed centralised management to optimize the energy of NoC routers. It exploits part of system properties like the AER task accesses model and the deadline slacks of timing-critical tasks to save energy accounting for real-time constraints. The approach is introduced through a control layer that is is developed on top of the existing NoC data layer to isolate energy-related messages than regular data transmissions as in contemporary commercial MPSoCs that use NoCs [26, 25, 151, 16].

Power-Aware NoC Controllers (PANCs) have been proposed through the NoC control layer to safely save energy on NoC routers based on their global knowledge of the system state. The safe combinations of multiple energy-savings schemes (*Power-Gating, Clock-Gating, and Dynamic Voltage and Frequency Scaling*) have been investigated by PANCs to efficiently manage various energy sources (leakage, clock-tree, and dynamic). PANCs functionality has been elaborated in Chapter 3. Moreover, formal worst-case analysis frameworks have been developed to upper bound the latency overhead of the energy control layer to account for timing constraints (presented in Chapter 4). Based on that, the permissible slacks of critical functions are extracted (at design time) and exploited by PANCs (at runtime) to monitor the NoC states, and thus applying the potential energy-savings – assuring temporal guarantees. The energy-aware NoC design has been introduced from concept to place and routed layout employing ASIC design flow, extracting efficient and accurate energy figures.

Furthermore, the concept of the energy control layer at the NoC-level has been extended to include degradation treatment at the system-level. For example, a foreseen core failure that indicates an imminent hazard occurrence in the system. To allow imminent hazards handling in mixed-criticality systems, flexible boundaries over the contemporary static ones between safety-critical and best-effort subsystems are enabled at runtime. To this end, distributed local managers cooperating in synergy and orchestrated by the system controller to reconfigure the system through container migration have been proposed. Besides, operation planning is supported, such that reconfiguration setups are planned ahead of time enabling predictability. As we pursue runtime adaptations for mixed-criticality systems, fundamental requirements derived from safety standards [81, 78] including isolation between safety-critical and best-effort subsystems, protected accesses to shared resources, temporal predictability, and deadlock-free reconfiguration have been satisfied. That is, the safety of the supported movable boundaries feature has been proved (as introduced in Chapter 5).

The approach is not limited to certain usecases nor architectural parameters. Rigorous simulation and formal analysis-based experiments have been conducted to respectively derive results for average and worst-case scenarios on various bench-

marks and realistic usecases. The results indicate significant improvements in NoC energy-savings and in treatment of system degradation for mixed-criticality systems improving dependability over the status quo. The presented approach could be also applied on diverse NoC architectural parameters (e.g., NoC size, topology, routing algorithm) as the actual implementation of the controllers is transparent to the underlying NoC architecture. The same goes for the processor type. That is, the approach could be easily ported to other platforms supporting other processor types (e.g., RISC-V or Leon3), as it is only required to copy the state of the processor registers from a failing core to a healthy one. The applicability of the approach in cyber-physical systems is very wide because mixed-criticality systems on many-core platforms are used in all major areas with dependability requirements from aerospace to medical applications, industry 4.0 or building automation. These applications share the continuous service requirements of the automotive applications while some of them require even never-down operations, such as a life-supporting pacemaker, or a smart grid.

There are some envisioned future outlines that provide a direction for future research regarding the dependable operation of MPSoCs. The current work provides the fundamental basis required to perform safety-compliant online reconfiguration in mixed-criticality systems employing homogeneous systems, but can generally be used for heterogeneous systems, as long as at least part of the cores can be replaced with a suitable mechanism. Also, future research work could extend the scope from one MPSoC to multiple ones. That is, larger distributed systems with heterogeneous MPSoCs could be investigated providing dependable computations and off-chip communications, supporting self-awareness in distributed systems.

Appendix A: List of Publications

Based on the work presented in this thesis, we provide in the following the list of all publications written by the author.

- Thawra Kadeed, Eberle A Rambo, and Rolf Ernst. Power and area evaluation of a fault-tolerant network-on-chip. In System-on-Chip Conference (SOCC), 30th IEEE International, pages 190-195, 2017.
  In this work, we evaluate the overhead of real-time networks-on-chip for ASICs in terms of area and power. We employ a power analysis framework and load profiles to provide accurate power figures. We analyse the power behavior in normal operation as well as under errors. This work is considered as a baseline evaluation employed later in this these to evaluate the NoC energy dissipation under a control layer.

- Thawra Kadeed, Sebastian Tobuschat, Adam Kostrzewa and Rolf Ernst. Safe and efficient power management of hard real-time networks-on-chip. Integration, the VLSI Journal, 2018.
  This article proposes a safe and enhanced approach for power-gating that allows a global and dynamic power management of networks-on-chip with temporal guarantees, i.e., all deadlines of critical tasks are met. It introduces the control layer to save power on the NoC data layer using multiple Power-Aware Network Controllers (PANCs). The latter apply knowledge of the global state of the system and exploit the slacks of timing-critical tasks to efficiently save power even at high NoCs utilizations. To safely apply the PANCs in hard real-time systems while meeting the deadlines, a formal worst-case timing analysis to upper bound PANCs' latency overhead is provided. The control layer achieves improved static power-savings over the related work where NoC energy is derived using ASIC design flow for both 65*nm* (UMCs) and 28*nm* (TSMCs) technologies. Also, the approach allows scalability inducing very small area overhead.

- Adam Kostrzewa, Thawra Kadeed, Borislav Nikolic, and Rolf Ernst. Supporting dynamic voltage and frequency scaling in networks-on-chip for hard real-time systems. In IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pages 125-135, 2018.

This work targets static and dynamic energy optimization for NoCs. It employs the control layer where PANCs perform dynamic voltage and frequency scaling at runtime. This is based on the number of concurrently active applications using protocol-based synchronization. The safety of the solution is assured by formal analysis and extensive experimental evaluation showing the improved energy-savings over the state-of-the-art static solutions.

- Armin Sadighi, Bryan Donyanavard, Thawra Kadeed, Kasra Moazzemi, Tiago Mück, Ahmed Nassar, Amir M. Rahmani, Thomas Wild, Nikil Dutt, Rolf Ernst, Andreas Herkersdorf and Fadi J. Kurdahi. Design Methodologies for Enabling Self-awareness in Autonomous Systems. In Proceedings of Design, Automation and Test in Europe Conference (DATE), 2018.
  This work deals with challenges and possible solutions for incorporating self-awareness principles in EDA design flows for autonomous systems. We present a holistic approach that enables self-awareness across the software/hardware stack, from systems-on-chip to systems-of-systems (autonomous car) contexts. We use the Information Processing Factory (IPF) metaphor as an exemplar to show how self-awareness can be achieved across multiple abstraction levels, and discuss new research challenges.

- Thawra Kadeed, Sebastian Tobuschat, and Rolf Ernst. Integrated energy control for hard real-time networks-on-chip. In IEEE Real-Time Systems Symposium (RTSS), pages 4-16, 2019.
  This work optimizes various energy dissipation sources: static energy, clock-tree energy, and dynamic energy for NoCs. To this end, PANCs explore the potential efficiency of integrating multiple energy-savings schemes in the face of the diversity of energy sources. PANCs feature two combinations of energy schemes: Power-Gating and Clock-Gating (*PG+CG*), and (*PG+CG+DVFS*). In addition, the temporal safety of the control layer is provided via formal worst-case timing analysis. Experimental evaluations demonstrate the efficient savings of various energy sources supporting scalability.

- Thawra Kadeed and Rolf Ernst. A comprehensive framework for energy management of hard real-time networks-on-chip. RTSS@ WORK 2019, page 7.
  This work extends the previous one to emphasize the comprehensive NoC energy control framework. The latter comprises three processing stages. The offline stage that supports PANCs with the extracted applications properties (e.g., slacks); PANCs stage to manage the NoC energy; and last stage extracts NoC energy-savings figures based on the observed traces and the baseline NoC energy.

- Thawra Kadeed and Rolf Ernst. Self-aware safety-critical systems-on-chip through an online system controller approach. In International Workshop on Cross-layer Resiliency (IWCR), 2019.
  This work introduces the principles of extending the concept of NoC control layer to system level to address the response to system early degradation by including treatment of imminent system failures. In the context of IPF project, we achieved these goals via supporting hierarchical reconfiguration managers (local managers supervised by a global one, the system controller) in abnormal behaviors (foreseen core failures). Such an approach allows predictability where system movements are analysable and controllable which enables temporal and functional verifications required by safety standards [81, 78, 49].

- Eberle A Rambo, Thawra Kadeed, Rolf Ernst, Minjun Seo, Fadi Kurdahi, Bryan Donyanavard, Caio Batista de Melo, Biswadip Maity, Kasra Moazzemi, Kenneth Stewart, et al. The information processing factory: a paradigm for life cycle management of dependable systems. In International Conference on Hardware/-Software Codesign and System Synthesis (CODES+ ISSS), pages 1-10, 2019.
  In this work, IPF applies principles inspired by factory management to master the complexity of future, highly integrated embedded systems and to provide continuous operation and optimization at runtime. The research work addresses the challenges of IPF and how to tackle them with a set of techniques: self-diagnosis for early detection of degradation and imminent failures combined with self-adaptation to meet performance and safety targets.

- Eberle Rambo, Bryan Donyanavard, Minjun Seo, Florian Maurer, Thawra Kadeed, Caio De Melo, Biswadip Maity, Anmol Surhonne, Andreas Herkersdorf, Fadi Kurdahi, Nikil Dutt, and Rolf Ernst. The information processing factory: Organization, terminology, and definitions. arXiv preprint arXiv:1907.01578, 2019.
  This report introduces the organization, terminology, and definitions of IPF. It describes IPF's five-layer hierarchical organization and a system configuration framework that enables self-awareness, self-diagnosis, self-organization, and self-optimization in mixed-criticality real-time embedded systems. It defines imminent hazards, which are threats to the system operation, that can be handled by IPF before they cause system failure.

- Eberle Rambo, Bryan Donyanavard, Minjun Seo, Florian Maurer, Thawra Kadeed, Caio De Melo, Biswadip Maity, Anmol Surhonne, Andreas Herkersdorf, Fadi Kurdahi, Nikil Dutt, and Rolf Ernst. The Self-Aware Information Processing Factory Paradigm for Mixed-Critical Multiprocessing. IEEE Transactions on Emerging Topics in Computing, 2020.

This article presents the IPF paradigm for mixed-criticality. We detail its 5-layer hierarchical organization and the system configuration framework that ensures that the strict requirements of the safety-critical functions are always met, while dynamically managing and optimizing the mixed-critical system at runtime. We illustrate the application of IPF in heterogeneous domains with two representative usecases (healthcare and automotive), investigate the use of IPF to achieve long-term dependability, and highlight open challenges.

- Thawra Kadeed and Rolf Ernst. Dynamic Energy Management of Mixed-Criticality Real-Time Networks-on-Chip. In Design, Automation and Test in Europe Conference (DATE), 2020.
  This work presents a predictable and safe energy management mechanism for NoC routers. The energy control layer through PANCs optimizes either individual energy source or multiple ones. That is, PANCs feature either *PG*, *CG*, or *DVFS*; or integrate these schemes on the same execution platform based on the user requirements and energy-savings granularities.

- Thawra Kadeed, Borislav Nikolic, and Rolf Ernst. Safe Online Reconfiguration of Mixed-Criticality Real-Time Systems. In Pacific Rim International Symposium on Dependable Computing (PRDC), Perth, Australia, 2020.
  This work presents and evaluates a safe online reconfiguration approach for many-core mixed-critical platforms. The approach enforces predictable system-wide reconfigurations through containers migration via enabling movable boundaries between safety-critical and best-effort subsystems. To achieve a safe move between the boundaries over the status quo, the approach satisfies fundamental isolation requirements and supports robust reconfiguration. To this end, formal analysis frameworks are supported. Also, it allows continuous real-time system operation as required in many application domains. The approach is particularly suited to provide proactive actions in the face of foreseen system failures improving availability without increased risk of failure during reconfiguration. Experimental evaluations indicate the improvements of the approach over the status quo allowing predictable online reconfigurations with tolerable latency overhead.

Bibliography

[1] https://semiengineering.com/the-race-to-autonomous-cars/.

[2] Laure Abdallah, Mathieu Jan, Jérôme Ermont, and Christian Fraboul. Reducing the contention experienced by real-time core-to-i/o flows over a tilera-like network on chip. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 86–96, 2016.

[3] Ankur Agarwal, Cyril Iskander, and Ravi Shankar. Survey of network on chip (noc) architectures & contributions. *Journal of engineering, Computing and Architecture*, 3(1):21–27, 2009.

[4] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *2009 IEEE international symposium on performance analysis of systems and software*, pages 33–42. IEEE, 2009.

[5] M. Amoretti. Modeling and Simulation of Network-on-Chip Systems with DEVS and DEUS. *The Scientific World*, 2014.

[6] Arteris. FlexNoC Interconnect IP. http://www.arteris.com/flexnoc. [accessed 22-June-2020], 2018.

[7] Seyyed Amir Asghari and et al. Enhancing transient fault tolerance in embedded systems through an os task level redundancy approach. *Future Generation Computer Systems*, 87:58–65, 2018.

[8] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[9] Jean-Luc Autran, P Roche, S Sauze, G Gasiot, Daniela Munteanu, P Loaiza, M Zampaolo, and J Borel. Altitude and underground real-time SER characterization of CMOS 65 nm SRAM. *IEEE Transactions on Nuclear Science*, 56(4):2258–2266, 2009.

[10] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[11] Philip Axer, Jonas Diemer, Mircea Negrean, Maurice Sebastian, Simon Schliecker, and Rolf Ernst. Mastering MPSoCs for Mixed-critical Applications. *Information and Media Technologies*, 6(4):1027–1052, 2011.

[12] Philip Axer, Maurice Sebastian, and Rolf Ernst. Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints. In *Proceedings of the international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, pages 149–158, 2011.

[13] Arnab Banerjee, Pascal T Wolkotte, Robert D Mullins, Simon W Moore, and Gerard JM Smit. An energy and performance exploration of network-on-chip architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(3):319–329, 2009.

[14] Arnab Banerjee, Pascal T Wolkotte, Robert D Mullins, Simon W Moore, and Gerard JM Smit. An energy and performance exploration of network-on-chip architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(3):319–329, 2009.

[15] Sanjoy Baruah, Haohan Li, and Leen Stougie. Towards the design of certifiable mixed-criticality systems. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–22, 2010.

[16] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, et al. Tile64-processor: A 64-core soc with mesh interconnect. In *IEEE International Solid-State Circuits Conference-Digest of Technical Papers*, pages 88–598, 2008.

[17] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. HNOCS: modular open-source simulator for heterogeneous NoCs. In *International Conference on Embedded Computer Systems (SAMOS)*, pages 51–57, 2012.

[18] Luca Benini and Giovanni De Micheli. Powering networks on chips: energy-efficient and reliable interconnect design for SoCs. In *Proceedings of the 14th international symposium on Systems synthesis*, pages 33–38, 2001.

[19] Luca Benini and Giovanni De Micheli. Networks on chips: A new SoC paradigm. *computer*, 35(1):70–78, 2002.

[20] D. Bertozzi, L. Benini, G. Micheli, and D. Error control schemes for on-chip communication links: the energy-reliability trade-off. *Transactions on CAD, IEEE*, 24(6), 2005.

[21] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[22] S. Borkar. Designing reliable systems from unreliable components:the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, November 2005.

[23] Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual design automation conference*, pages 746–749, 2007.

[24] Shekhar Borkar. Future of interconnect fabric: a contrarian view. In *Proceedings of the 12th ACM/IEEE international workshop on System level interconnect prediction*, pages 1–2, 2010.

[25] Marc Boyer, Benoît Dupont de Dinechin, Amaury Graillat, and Lionel Havet. Computing routes and delay bounds for the network-on-chip of the Kalray MPPA2 processor. 2018.

[26] Marc Boyer, Amaury Graillat, Benoît Dupont de Dinechin, and Jörn Migge. Comparing strategies to bound the latencies of the mppa network-on-chipi. 2019.

[27] Alan Burns and Wellings Andy. *Real-Time Systems And Programming Languages*. Addison-Wesley, Reading, MA, 2001.

[28] Alan Burns and Robert Davis. Mixed criticality systems-a review, 9th edition. *Department of Computer Science, University of York, Tech. Rep*, 2017.

[29] Alan Burns and Robert I Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys (CSUR)*, (6):82, 2018.

[30] Alan Burns and Andy Wellings. Real-time systems and programming languages: Ada, real-time java and c/real-time posix. 2009.

[31] Jordi Cardona, Carles Hernandez, Enrico Mezzetti, Jaume Abella, and Francisco J Cazorla. NoCo: ILP-Based Worst-Case Contention Estimation for Mesh Real-Time Manycores. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 265–276, 2018.

[32] MR Casu, MK Yadav, and M Zamboni. Power-gating technique for network-on-chip buffers. *Electronics Letters*, 49(23):1438–1440, 2013.

[33] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. Cycle-accurate network on chip simulation with noxim. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 27(1):1–25, 2016.

[34] R. Chadha and J. Bhasker. *An ASIC Low Power Primer, Analysis, Techniques and Specification*. in Springer-Verlag, New York, 2013.

[35] Santanu Chattopadhyay and Santanu Kundu. *Network-on-Chip: The Next Generation of System-on-Chip Integration*. CRC press, 2014.

[36] Shubhangi D Chawade, Mahendra A Gaikwad, and Rajendra M Patrikar. Review of xy routing algorithm for network-on-chip architecture. *International Journal of Computer Applications*, 43(21):975–8887, 2012.

[37] Tiberiu Chelcea and Steven M Nowick. A low-latency FIFO for mixed-clock systems. In *Proceedings IEEE Computer Society Workshop on VLSI 2000. System Design for a System-on-Chip Era*, pages 119–126, 2000.

[38] Lizhong Chen and Timothy M Pinkston. Nord: Node-router decoupling for effective power-gating of on-chip routers. In *Microarchitecture (MICRO), 45th Annual IEEE/ACM International Symposium on*, pages 270–281, 2012.

[39] Lizhong Chen, Di Zhu, Massoud Pedram, and Timothy M Pinkston. Power punch: Towards non-blocking power-gating of noc routers. In *High Performance Computer Architecture (HPCA), IEEE 21st International Symposium on*, pages 378–389, 2015.

[40] X. Chen et al. In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches. In *NOCS*, 2012.

[41] Érika Cota, Marcelo Soares Lubaszewski, and Alexandre de Morais Amory. Reliability, availability and serviceability of networks-on-chip. 2012.

[42] Antonio Augusto da Fontoura, Francisco Assis Moreira do Nascimento, Simin Nadjm-Tehrani, and Edison Pignaton de Freitas. Timing assurance of avionic reconfiguration schemes using formal analysis. *IEEE Transactions on Aerospace and Electronic Systems*, 2019.

[43] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[44] Reetuparna Das, Satish Narayanasamy, Sudhir K Satpathy, and Ronald G Dreslinski. Catnap: Energy proportional multiple network-on-chip. *ACM SIGARCH Computer Architecture News*, 41(3):320–331, 2013.

[45] Bhavya K Daya, Owen Chen, Suvinay Subramanian, Woo-Cheol Kwon, Sunghyun Park, Tushar Krishna, Jim Holt, Anantha P Chandrakasan, and Li-Shiuan Peh. SCORPIO: a 36-core research chip demonstrating snoopy coherence on a scalable mesh NoC with in-network ordering. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 25–36. IEEE Press, 2014.

[46] Benoît Dupont De Dinechin, Duco Van Amstel, Marc Poulhiès, and Guillaume Lager. Time-critical computing on a single-chip massively parallel processor. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2014.

[47] Bjorn De Sutter, Praveen Raghavan, and Andy Lambrechts. Coarse-grained reconfigurable array architectures. In *Handbook of signal processing systems*, pages 427–472. Springer, 2019.

[48] DO-178B: software considerations in airborne systems and equipment certification, 2009.

[49] DO-254: Design assurance guidance for airborne electronic hardware. RTCA Incorporated, 2000.

[50] Guy Durrieu, Madeleine Faugere, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. Predictable flight management system implementation on a multicore processor. In *Embedded Real Time Software (ERTS)*, 2014.

[51] Guy Durrieu and Claire Pagetti. GRec: Automatic computation of reconfiguration graphs for multi-core platforms. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5):1–24, 2019.

[52] Ashaf El-Antably, Olivier Gruber, Frederic Rousseau, and Nicolas Fournel. Transparent and portable agent based task migration for data-flow applications on multi-tiled architectures. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 183–192, 2015.

[53] Rolf Ernst. *Electronic Systems Design*. Institute of Computer and Communication Engineering Lectures, 2017.

[54] Rolf Ernst and Marco Di Natale. Mixed criticality systems - a history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[55] José V Escamilla, José Flich, and Mario R Casu. ICARO-PAPM: Congestion Management with Selective Queue Power-Gating. In *International Conference on High Performance Computing & Simulation (HPCS)*, pages 259–266, 2017.

[56] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, June 2011.

[57] Chris Fallin, Chris Craik, and Onur Mutlu. Chipper: A low-complexity bufferless deflection router. In *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*, pages 144–155, 2011.

[58] Hossein Farrokhbakht, Mohammadkazem Taram, Behnam Khaleghi, and Shaahin Hessabi. Toot: an efficient and scalable power-gating method for noc routers. In *Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, 2016.

[59] Gerhard Fohler, Gautam Gala, Daniel Gracia Pérez, and Claire Pagetti. Evaluation of dreams resource management solutions on a mixed-critical demonstrator. In *ERTS*, 2018.

[60] Stephanie Friederich, Marco Neber, and Jürgen Becker. Power management controller for online power saving in network-on-chips. In *EEE 10th*

International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), pages 109–116, 2016.

[61] Rémi Gaillard. *Single Event Effects: Mechanisms and Classification*, pages 27–54. Springer US, Boston, MA, 2011.

[62] Laurent Gantel, Salah Layouni, Mohamed El Amine Benkhelifa, François Verdier, and Stéphanie Chauvet. Multiprocessor task migration implementation in a reconfigurable platform. In *International Conference on Reconfigurable Computing and FPGAs*, pages 362–367, 2009.

[63] Zana Ghaderi, Ayed Alqahtani, and Nader Bagherzadeh. Online monitoring and adaptive routing for aging mitigation in NoCs. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 67–72. European Design and Automation Association, 2017.

[64] Paul Gratz, Changkyu Kim, Robert McDonald, Stephen W Keckler, and Doug Burger. Implementation and evaluation of on-chip network architectures. In *2006 International Conference on Computer Design*, pages 477–484. IEEE, 2006.

[65] Cristian Grecu, Andre Ivanov, Res Saleh, Egor S Sogomonyan, and Partha Pratim Pande. On-line fault detection and location for NoC interconnects. In *12th International On-Line Testing Symposium (IOLTS)*, 2006.

[66] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE International Workshop on Workload Characterization, WWC-4*, 2001.

[67] Khaled A Helal, Sameh Attia, Tawfik Ismail, and Hassan Mostafa. Comparative review of NoCs in the context of ASICs and FPGAs. In *IEEE International Symposium on Circuits and Systems (ISCAS).*, 2015.

[68] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis–the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

[69] J. Henkel, H. Khdr, S. Pagani, and M. Shafique. New trends in dark silicon. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[70] Jörg Henkel, Lars Bauer, Joachim Becker, Oliver Bringmann, Uwe Brinkschulte, Samarjit Chakraborty, Michael Engel, Rolf Ernst, Hermann Härtig, Lars Hedrich, et al. Design and architectures for dependable embedded systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 69–78. ACM, 2011.

[71] John L. Hennessy and David A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6th edition, 2017.

[72] Andreas Herkersdorf, Hananeh Aliee, Michael Engel, Michael Glaß, Christina Gimmler-Dumont, Jörg Henkel, Veit B Kleeberger, Michael A Kochte, Johannes M Kühn, Daniel Mueller-Gritschneder, et al. Resilience Articulation Point (RAP): Cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectronics Reliability*, 54(6–7):1066–1074, 2014.

[73] Salma Hesham, Jens Rettkowski, and et al. Survey on real-time networks-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1500–1517, May 2017.

[74] Robert Hesse and Natalie Enright Jerger. Improving DVFS in NoCs with coherence prediction. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 24. ACM, 2015.

[75] Robert Hesse and Natalie Enright Jerger. Improving DVFS in NoCs with Coherence Prediction. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, NOCS, 2015.

[76] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, 2007.

[77] Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. Microarchitectural techniques for power gating of execution units. In *Proceedings of the international symposium on Low power electronics and design*, pages 32–37. ACM, 2004.

[78] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, ed.2.0. International Electrotechnical Commission, 2010.

[79] L. S. Indrusiak, A. Burns, and B. Nikolić. Buffer-aware bounds to multi-point progressive blocking in priority-preemptive NoCs. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 219–224, March 2018.

[80] Leandro Soares Indrusiak. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of systems architecture*, 60(7):553–561, 2014.

[81] ISO 26262: Road vehicles – functional safety. International Standards Organization, 2018.

[82] Wooyoung Jang and David Z Pan. A voltage-frequency island aware energy optimization framework for networks-on-chip. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(3):420–432, 2011.

[83] A. Jantsch, N. Dutt, and A. M. Rahmani. Self-awareness in systems on chip– a survey. *IEEE Design Test*, 34(6):8–26, Dec 2017.

[84] Nan Jiang, Daniel U Becker, George Michelogiannakis, James Balfour, Brian Towles, David E Shaw, John Kim, and William J Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 86–96. IEEE, 2013.

[85] Thawra Kadeed and Rolf Ernst. A comprehensive framework for energy management of hard real-time networks-on-chip. *RTSS@ WORK 2019*, page 7.

[86] Thawra Kadeed and Rolf Ernst. Self-aware safety-critical systems-on-chip through an online system controller approach. In *International Workshop on Cross-layer Resiliency (IWCR)*, 2019.

[87] Thawra Kadeed and Rolf Ernst. Dynamic Energy Management of Mixed-Criticality Real-Time Networks-on-Chip. In *Design, Automation and Test in Europe Conference (DATE)*, 2020.

[88] Thawra Kadeed, Borislav Nikolic, and Rolf Ernst. Safe Online Reconfiguration of Mixed-Criticality Real-Time Systems. In *Pacific Rim International Symposium on Dependable Computing (PRDC)*, Perth, Australia, 2020.

[89] Thawra Kadeed, Eberle A Rambo, and Rolf Ernst. Power and area evaluation of a fault-tolerant network-on-chip. In *System-on-Chip Conference (SOCC), 30th IEEE International*, pages 190–195, 2017.

[90] Thawra Kadeed, Sebastian Tobuschat, and Rolf Ernst. Integrated energy control for hard real-time networks-on-chip. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 4–16, 2019.

[91] Thawra Kadeed, Sebastian Tobuschat, Adam Kostrzewa, and Rolf Ernst. Safe and efficient power management of hard real-time networks-on-chip. *Integration, the VLSI Journal*, 2018.

[92] Nikolay Krasimirov Kavaldjiev and Gerardus Johannes Maria Smit. *A survey of efficient on-chip communications for soc*. Centre for Telematics and Information Technology, University of Twente, 2003.

[93] Cédric Killian, Camel Tanougast, Fabrice Monteiro, and Abbas Dandache. Smart reliable network-on-chip. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 242–255, 2014.

[94] Ryan Gary Kim, Wonje Choi, Zhuo Chen, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Imitation learning for dynamic VFI control in large-scale manycore systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(9):2458–2471, 2017.

[95] Ryan Gary Kim, Wonje Choi, Guangshuo Liu, Ehsan Mohandesi, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Wireless NoC for VFI-enabled multicore chip design: Performance evaluation and design trade-offs. *IEEE Transactions on Computers*, 65(4):1323–1336, 2015.

[96] Andre Kohn, Rolf Schneider, Antonio Vilela, Andre Roger, and Udo Dannebaum. Architectural concepts for fail-operational automotive systems. Technical report, SAE Technical Paper, 2016.

[97] Adam Kostrzewa. *Achieving Performance in Networks-On-Chip for Real-Time Systems*. PhD thesis, 2018.

[98] Adam Kostrzewa, Thawra Kadeed, Borislav Nikolić, and Rolf Ernst. Supporting dynamic voltage and frequency scaling in networks-on-chip for

hard real-time systems. In *IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 125–135, 2018.

[99] Adam Kostrzewa, Selma Saidi, Leonardo Ecco, and Rolf Ernst. Dynamic admission control for real-time networks-on-chips. In *Design Automation Conference (ASP-DAC), 21st Asia and South Pacific*, pages 719–724. IEEE, 2016.

[100] Adam Kostrzewa, Selma Saidi, and Rolf Ernst. Dynamic control for mixed-critical networks-on-chip. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 317–326, 2015.

[101] Adam Kostrzewa, Sebastian Tobuschat, and Rolf Ernst. Self-aware network-on-chip control in real-time systems. *IEEE Design & Test*, 35(5):19–27, 2017.

[102] Samuel Kounev, Peter Lewis, Kirstie L. Bellman, Nelly Bencomo, Javier Camara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger Giese, Sebastian Götz, Paola Inverardi, Jeffrey O. Kephart, and Andrea Zisman. *The Notion of Self-aware Computing*. Springer International Publishing, Cham, 2017.

[103] Harekrishna Kumar, Anjan Kumar, and Vinay Kumar Deolia. Enabling concurrent clock and power gating in 32 bit rom. In *9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6, 2018.

[104] J Jean Labrosse. *MicroC/OS-II the real-time kernel*. CMP, New York, 2002.

[105] Evelyn Landman, Shai Cohen, Noam Brousard, Raanan Gewirtzman, Inbar Weintrob, Eyal Fayne, Yahel David, Yuval Bonen, Omer Niv, Shai Tzroia, et al. Degradation monitoring—-from a vision to reality. In *IEEE International Reliability Physics Symposium (IRPS)*, pages 1–4, 2019.

[106] Fan Liu, Ajit Narayanan, and Quan Bai. Real-time systems. 2000.

[107] Meng Liu, Matthias Becker, Moris Behnam, and Thomas Nolte. A tighter recursive calculus to compute the worst case traversal time of real-time traffic over NoCs. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pages 275–282, 2017.

[108] Yanchen Long, Zhonghai Lu, and Haibin Shen. Composable worst-case delay bound analysis using network calculus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(3):705–709, 2018.

[109] Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. Nnse: Nostrum network-on-chip simulation environment. In *Swedish system-on-chip conference*, 2005.

[110] Enrico Macii, Leticia Bolzani, Andrea Calimera, Alberto Macii, and Massimo Poncino. Integrating clock gating and power gating for combined dynamic and leakage power optimization in digital cmos circuits. In *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, pages 298–303, 2008.

[111] Peter. Marwedel. *EMBEDDED SYSTEM DESIGN: Embedded Systems Foundations of Cyber-physical Systems, and The Internet of Things*. springer, 2019.

[112] Hiroki Matsutani, Michihiro Koibuchi, Daihan Wang, and Hideharu Amano. Run-time power gating of on-chip routers using look-ahead routing. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 55–60. IEEE Computer Society Press, 2008.

[113] Nick McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM transactions on networking*, 7(2):188–201, 1999.

[114] Mischa Möstl and et al. Self-adaptation for availability in cpu-fpga systems under soft errors. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 9–16, 2019.

[115] Mullins and Robert. Minimising dynamic power consumption in on-chip networks. In *System-on-Chip, International Symposium on. IEEE*, pages 1–4, 2006.

[116] Peter Munk and et al. Position paper: Real-time task migration on many-core processors. In *Proceedings ARCS 2015-The 28th International Conference on Architecture of Computing Systems.*, pages 1–4. VDE, 2015.

[117] S. Murali, Th. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and De Micheli G. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test of Computers*, 22(5):434–442, 2005.

[118] Abhishek Nag, Subhajit Das, and Sambhu Nath Pradhan. Low-power fsm synthesis based on automated power and clock gating technique. *Journal of Circuits, Systems and Computers*, page 1920003, 2018.

[119] Nasim Nasirian, Reza Soosahabi, and Magdy Bayoumi. Traffic-aware power-gating scheme for network-on-chip routers. In *Circuits and Systems Conference (DCAS), IEEE Dallas*, pages 1–4, 2016.

[120] Netspeed. Netspeed Gemini SoC Interconnect. `http://netspeedsystems.com/products/gemini/`. [accessed 22-June-2020], 2018.

[121] Netspeed. Netspeed Orion SoC Interconnect. `http://netspeedsystems.com/products/orion/`. [accessed 22-June-2020], 2018.

[122] Thanh Dinh Ngo and et al. Move based algorithm for runtime mapping of dataflow actors on heterogeneous mpsocs. *Journal of Signal Processing Systems*, 87(1):63–80, 2017.

[123] Borislav Nikolic, Robin Hofmann, and Rolf Ernst. Slot-Based Transmission Protocol for Real-Time NoCs - SBT-NoC. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, pages 26:1–26:22, 2019.

[124] Umit Y Ogras, Radu Marculescu, Puru Choudhary, and Diana Marculescu. Voltage-frequency island partitioning for GALS-based networks-on-chip. In *Proceedings of the 44th annual Design Automation Conference*, pages 110–115, 2007.

[125] Ritesh Parikh, Reetuparna Das, and Valeria Bertacco. Power-aware NoCs through routing and topology reconfiguration. In *Design Automation Conference (DAC), 51st ACM/EDAC/IEEE*, pages 1–6, 2014.

[126] A. Patooghy and S. G. Miremadi. XYX: A Power & Performance Efficient Fault-Tolerant Routing Algorithm for Network on Chip. In *17th Euromicro International Conference on Parallel, Distributed and Network-based Processing. IEEE*, 2009.

[127] Behnaz Pourmohseni, Stefan Wildermann, Michael Glaß, and Jürgen Teich. Hard real-time application mapping reconfiguration for noc-based many-core systems. *Real-Time Systems*, pages 1–37, 2019.

[128] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. Methods for fault tolerance in networks on chip. *ACM Comput. Surv*, 44:1–35, 2012.

[129] Amir M Rahmani, Bryan Donyanavard, Tiago Mück, Kasra Moazzemi, Axel Jantsch, Onur Mutlu, and Nikil Dutt. Spectr: Formal supervisory control and coordination for many-core systems resource management. *ACM SIGPLAN Notices*, 53(2):169–183, 2018.

[130] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst. Designing Networks-on-Chip for High Assurance Real-Time Systems. In *Pacific Rim International Symposium on Dependable Computing (PRDC)*, Christchurch, New Zealand, 2017.

[131] Eberle Rambo, Bryan Donyanavard, Minjun Seo, Florian Maurer, Thawra Kadeed, Caio De Melo, Biswadip Maity, Anmol Surhonne, Andreas Herkersdorf, Fadi Kurdahi, Nikil Dutt, and Rolf Ernst. The self-aware information processing factory paradigm for mixed-critical multiprocessing. *IEEE Transactions on Emerging Topics in Computing*, 2020.

[132] Eberle A Rambo, Bryan Donyanavard, Minjun Seo, Florian Maurer, Thawra Kadeed, Caio B de Melo, Biswadip Maity, Anmol Surhonne, Andreas Herkersdorf, Fadi Kurdahi, et al. The information processing factory: Organization, terminology, and definitions. *arXiv preprint arXiv:1907.01578*, 2019.

[133] Eberle A. Rambo and Rolf Ernst. Replica-Aware Co-Scheduling for Mixed-Criticality. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, volume 76, pages 20:1–20:20, 2017.

[134] Eberle A Rambo, Robin Hapka, and Rolf Ernst. A gem5 multi-os mixed-critical many-core simulation model for self-aware systems. In *Real-Time Systems Symposium (RTSS)*, 2019.

[135] Eberle A Rambo, Thawra Kadeed, Rolf Ernst, Minjun Seo, Fadi Kurdahi, Bryan Donyanavard, Caio Batista de Melo, Biswadip Maity, Kasra Moazzemi, Kenneth Stewart, et al. The information processing factory: a paradigm for life cycle management of dependable systems. In *2019 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–10, 2019.

[136] Eberle A Rambo, Yunsheng Shang, and Rolf Ernst. Providing integrity in real-time networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(8):1907–1920, 2019.

[137] Ismael Ripoll and Rafael Ballester-Ripoll. Period selection for minimal hyperperiod in periodic task systems. *IEEE Transactions on Computers*, 62(9):1813–1822, 2013.

[138] Armin Sadighi, Bryan Donyanavard, Thawra Kadeed, and et al. Design methodologies for enabling self-awareness in autonomous systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1532–1537, 2018.

[139] Siva Satyendra Sahoo, Bharadwaj Veeravalli, and Akash Kumar. A hybrid agent-based design methodology for dynamic cross-layer reliability in heterogeneous embedded systems. In *55th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2019.

[140] Luca Schenato, Bruno Sinopoli, Massimo Franceschetti, Kameshwar Poolla, and S Shankar Sastry. Foundations of control and estimation over lossy networks. *Proceedings of the IEEE*, 95(1):163–187, 2007.

[141] Johannes Schlatow, Mischa Möstl, Rolf Ernst, Marcus Nolte, Inga Jatzkowski, Markus Maurer, Christian Herber, and Andreas Herkersdorf. Self-awareness in autonomous automotive systems. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 1050–1055. European Design and Automation Association, 2017.

[142] Minjun Seo and Fadi Kurdahi. Efficient tracing methodology using automata processor. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–18, 2019.

[143] Mingoo Seok, Peter R Kinget, Teng Yang, Jiangyi Li, and Doyun Kim. Recent advances in in-situ and in-field aging monitoring and compensation for integrated circuits. In *IEEE International Reliability Physics Symposium (IRPS)*, pages 5C–1, 2018.

[144] Zheng Shi and Alan Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Networks-on-Chip, NoCS Second ACM/IEEE International Symposium on*, pages 161–170. IEEE, 2008.

[145] Zheng Shi, Alan Burns, and Leandro Soares Indrusiak. Schedulability analysis for real time on-chip communication with wormhole switching. *Innovations in embedded and real-time systems engineering for communication*, page 198, 2012.

[146] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–10, 2013.

[147] Jarpula Srikanth. An approach for low leakage power by power gating stack technique. *IJITR*, 4(6):4579–4582, 2016.

[148] John A. Stankovic, Krithi Ramamritham, and Marco Spuri. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[149] John A Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*, volume 460. Springer Science & Business Media, 2012.

[150] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE micro*, 22(2):25–35, 2002.

[151] Tilera. Tile processor architecture overview for the tilepro series. 2011.

[152] Sebastian Tobuschat. *Predictable and Runtime-Adaptable Network-On-Chip for Mixed-critical Real-time Systems*. PhD thesis, TU Braunschweig, 2019.

[153] Sebastian Tobuschat, Philip Axer, Rolf Ernst, and Jonas Diemer. IDAMC: A noc for mixed criticality systems. In *19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 149–156, 2013.

[154] Sebastian Tobuschat and Rolf Ernst. Efficient Latency Guarantees for Mixed-Criticality Networks-on-Chip. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Pittsburgh, PA, USA*, 2017.

[155] Sebastian Tobuschat and Rolf Ernst. Real-time communication analysis for networks-on-chip with backpressure. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 590–595. European Design and Automation Association, 2017.

[156] Sebastian Tobuschat, Rolf Ernst, Arne Hamann, and Dirk Ziegenbein. System-level timing feasibility test for cyber-physical automotive systems. In *11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2016.

[157] Sebastian Tobuschat, Adam Kostrzewa, Falco K Bapp, and Christoph Dropmann. Online monitoring for safety-critical multicore systems. *it-Information Technology*, 59(5):215–222, 2017.

[158] Sriram R Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Arvind Singh, Tiju Jacob, Shailendra Jain, et al. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of solid-state circuits*, 43(1):29–41, 2008.

[159] H. Wang and et al. Orion: a power-performance simulator for interconnection networks. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture,(MICRO-35)*, 2002.

[160] Hangsheng Wang, Li-Shiuan Peh, and Sharad Malik. Power-driven design of router microarchitectures in on-chip networks. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 105. IEEE Computer Society, 2003.

[161] X. Wang et al. Fine-grained runtime power budgeting for networks-on-chip. In *ASPDAC*, 2015.

[162] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE micro*, 27(5):15–31, 2007.

[163] Jae-Yeon Won, Xi Chen, Paul Gratz, Jiang Hu, and Vassos Soteriou. Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management. In *HPCA*, 2014.

[164] Yuejian Wu, Sandy Thomson, and et al. Free Razor: A Novel Voltage Scaling Low-Power Technique for Large SoC Designs. *in IEEE Transactions on VLSI Systems*, 23(11):2431–2437, 2015.

[165] Qin Xiong, Zhonghai Lu, Fei Wu, and Changsheng Xie. Real-time analysis for wormhole noc: Revisited and revised. In *International Great Lakes Symposium on VLSI (GLSVLSI)*, pages 75–80, 2016.

[166] T. T. Ye, G. Micheli, D., and L. Benini. Analysis of power consumption on switch fabrics in network routers. In *Proceedings of the 39th annual Design Automation Conference. ACM*, 2002.

[167] Jia Zhan, Nikolay Stoimenov, Jin Ouyang, Lothar Thiele, Vijaykrishnan Narayanan, and Yuan Xie. Designing energy-efficient NoC for real-time embedded systems through slack optimization. In *Proceedings of the 50th Annual Design Automation Conference*, page 37. ACM, 2013.

[168] Jia Zhan, Nikolay Stoimenov, Jin Ouyang, Lothar Thiele, Vijaykrishnan Narayanan, and Yuan Xie. Optimizing the NoC slack through voltage and frequency scaling in hard real-time embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(11):1632–1643, 2014.

[169] Dirk Ziegenbein and Arne Hamann. Timing-aware control software design for automotive systems. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[170] Davide Zoni and William Fornaciari. Modeling DVFS and power-gating actuators for cycle-accurate NoC-based simulators. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 12(3):27, 2015.

Glossary

mixed-criticality system A system that adopts different criticality levels of applications on the same execution platform, where some applications are more important to the overall value than others [54, 28].. 11

safety Freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment [81].. 11

quality of service The collective effect of service performances, which determine the degree of satisfaction of a user of the service. These characteristic performances may, for example, relate to: transmission quality, timing, failures, fault frequency and duration [152].. 12

tile A tile consists of one or more compute elements (e.g., core), a network interface of network-on-chip, local memory (cache or SPM), and a bus connecting these components. 12

network-on-chip The backbone of the multiprocessor system-on-chip, connecting all tiles to each other, to the memory and to the peripherals [5].. 12

timing-critical A timing-critical system is a system where all applications have firm deadlines that must be met.. 12

dependability of a computing system is the ability to deliver service that can justifiably be trusted [10] 17

latency Time required to deliver a unit of data (usually a packet or message) through the network, measured from its injection at the source to its ejection at the destination [43].. 19

predictability The degree to which a correct prediction or forecast of system's state can be made either qualitatively or quantitatively. Formal analysis

methods can be employed to predtict the worst-case bahavior of a system or component.. 20, 172

system controller A centralized component, responsible for controlling the system under severe changes (e.g. imminent core failure) to bring the system back to its normal operation.. 21

deadline slack Time budget between the task's deadline and its worst-case response time.. 35

power-aware network controller A network-on-chip controller that grants the senders the netwrok access based on their timing requirements and the routers states (energy modes) to efficiently optimize the network energy.. 37

imminent hazard An increased risk of future errors that can lead to system failure and, therewith, a hazard.. 46

container It encapsulates a software stack required for the execution of the respective workload, including runtime environment, and an operating system.. 47

SC container Safety-Critical workload specific container.. 47

BE container Best-Effort workload specific container.. 48

planner A component responsible for the long-term planning of the system. It creates sets of plans/next operating points that will be chosen and executed upon imminent hazard occurrences.. 48

operating point An operating point is a specific configuration for the containers, resources, containers' mapping and resources' allocation.. 49

SC zone A Safety-Critical zone corresonds to the set of resources allocated to the safety-critical workload execution.. 49

BE zone A best-effort zone corresponds to the set of resources allocated to the best-effort workload execution.. 49

worst-case response time Maximum potentoal time between releasing a job and its completion.. 96

safety-critical A safety-critical system is a system where a malfunction can cause an "unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment".. 127

plan The potential hazard together with the corresponding configuration (including the best-effort tiles to which we can migrate the safety-critical container) are called a plan.. 143

next operating point See the definition of plan. 143

Acronyms

AER Acquisition-Execution-Restitution

Ack Acknowledgement

ASIL automotive safety integrity level

ADAS Advanced Driver Assistance Systems

ASIC Application-Specific Integrated Circuit

BET Break-Even Time

BE Best-Effort

BF Body Flit

BEC Best-Effort Controller

CG Clock-Gating

CMPs Chip-MultiProcessors

CTC Critical Tile Controller

COP Current Operating Point

CPA compositional performance analysis

DVFS Dynamic Voltage and Frequency Scaling

DMR Dual Modular Redundancy

DMA Direct Memory Access

DC Design Compiler

ECU Electronic Control Unit

EM Energy Mode

FADEC Full Authority Digital Engine Control

FPGA Field-Programmable Gate Array

FLIT Flow Control Unit

FMEA Failure Mode and Effects Analysis

HF Head Flit

IP Intellectual Property

IDAMC Integrated Dependable Architecture for Many Cores

IEC International Electrotechnical Commission

ISO International Standards Organization

IPF Information Processing Factory

IH Imminent Hazard

MPSoC Multiprocessor System-on-Chip

MCS Mixed-criticality system

NoC Network-on-Chip

NI network interface

NIC Network Interface Controller

NOP Next Operating Point

Opt-PANC Optimized PANC

OS Operating System

OP Operating Point

PANC Power-Aware Network Controller

PG Power-Gating

QoS Quality-of-Service

Req Request

Rel Release

RTL Register Transfer Level

RM Resource Manager

RTOS Real-Time Operating System

RTE Runtime Environment

SoC System-on-Chip

SPP Static-Priority Preemptive

SPNP Static Priority Non Preemptive

SC Safety-Critical

SyC System Controller

TMR Triple Modular Redundancy

TSMC Taiwan Semiconductor Manufacturing Company

TF Tail Flit

UMC United Microelectronics Corporation

VC Virtual Channel

VCAC Virtual Channel Access Controller

VFI Voltage-Frequency-Islands

WCET Worst-Case Execution Time